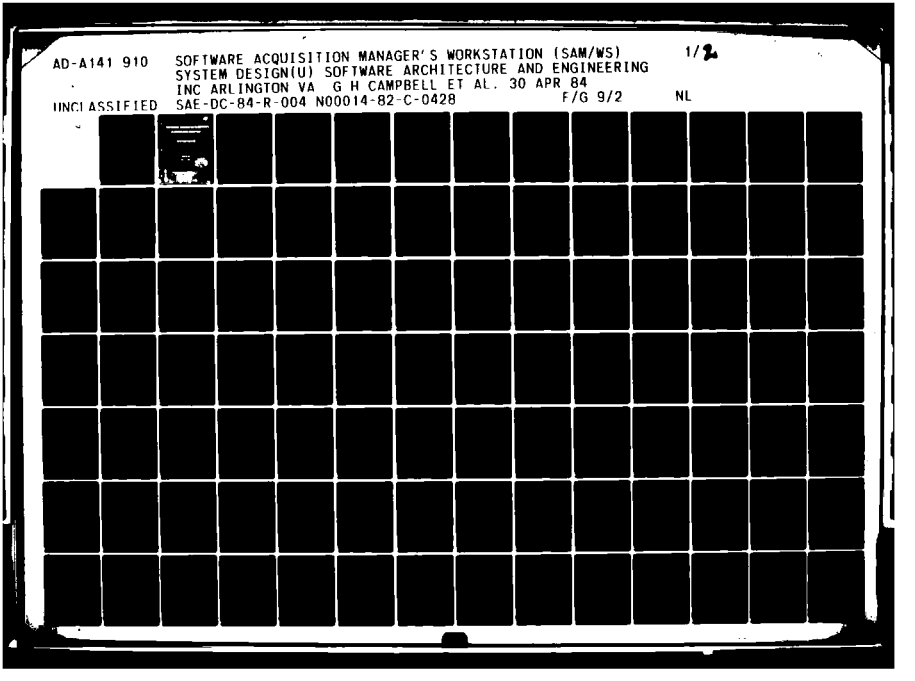
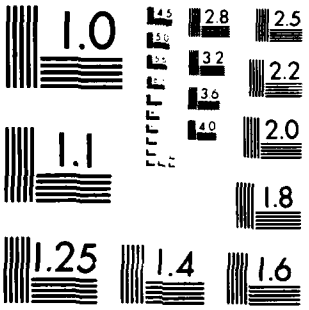
AD-A141 910 SOFTWARE ACQUISITION MANAGER'S WORKSTATION (SAM/WS) SYSTEM DESIGN(U) SOFTWARE ARCHITECTURE AND ENGINEERING INC ARLINGTON VA G H CAMPBELL ET AL. 30 APR 84 1/2

UNCLASSIFIED SAE-DC-84-R-004 N00014-82-C-0428 F/G 9/2 NL





Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

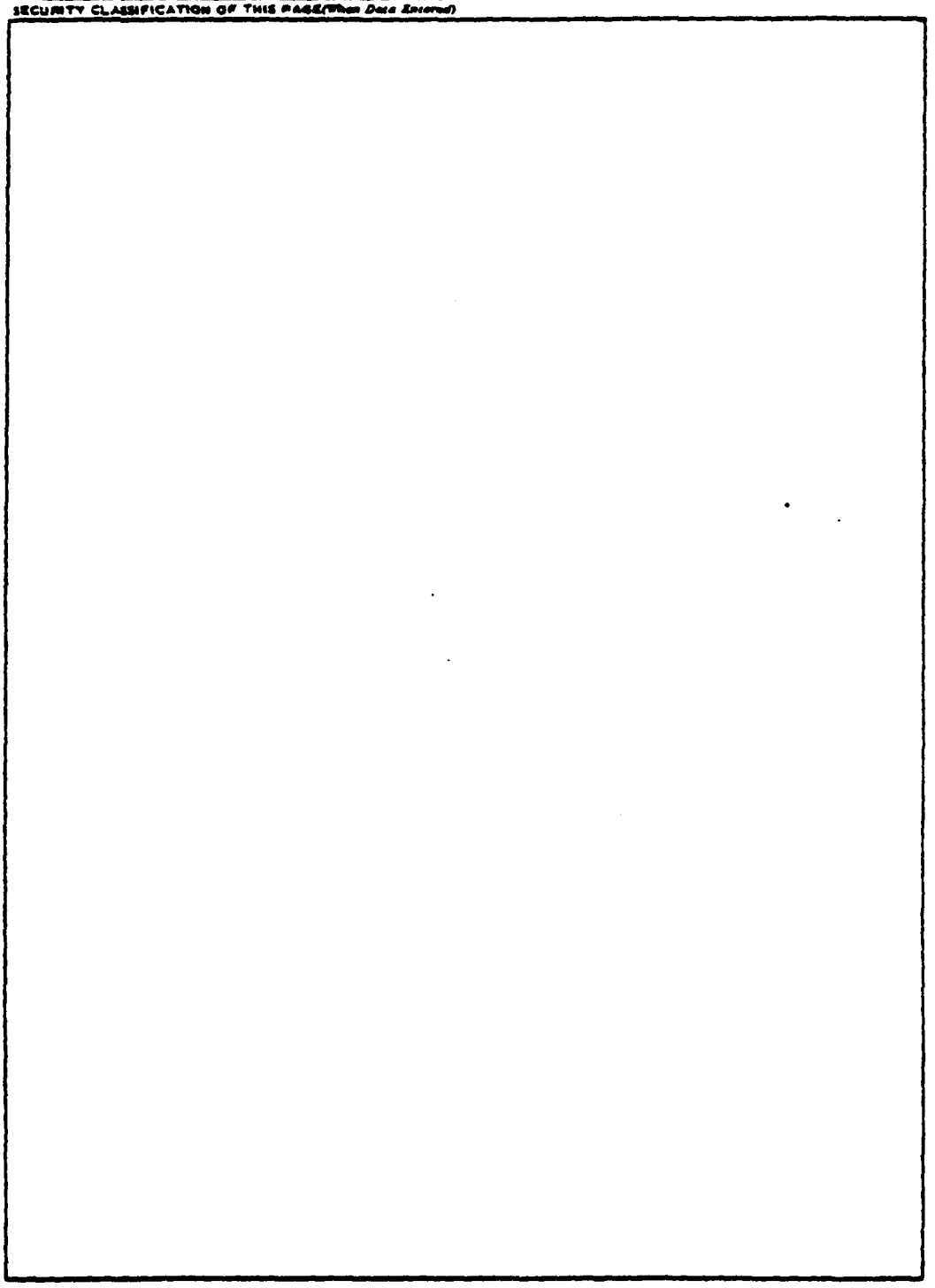
REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: N00014-82-C-0428	2. GOVT ACCESSION NO.: AD-A141 912
3. RECIPIENT'S CATALOG NUMBER:	
4. TITLE (and Subtitle): Software Acquisition Manager's Workstation (SAM/WS) System Design	5. TYPE OF REPORT & PERIOD COVERED: Final Report Seq No. A015
	6. PERFORMING ORG. REPORT NUMBER: SAE-DC-84-R-004
7. AUTHOR(s): G. H. Campbell, J. W. Sapp	8. CONTRACT OR GRANT NUMBER(s): N00014-82-C-0428
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Software A&E, Inc. 1401 Wilson Blvd., Arlington, VA 22209	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS:
11. CONTROLLING OFFICE NAME AND ADDRESS: Department of the Navy Office of Naval Research, Arlington, VA	12. REPORT DATE: April 30, 1984
	13. NUMBER OF PAGES: 180
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): OSD STARS Program Office Rm 3D-139 (400 Army Navy Drive) The Pentagon, Washington, DC 20301	15. SECURITY CLASS. (of this report): Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:
16. DISTRIBUTION STATEMENT (of this Report): This document has been approved for public release and sale; its distribution is unlimited.	
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report):	
18. SUPPLEMENTARY NOTES:	
19. KEY WORDS (Continue on reverse side if necessary and identify by block number): software acquisition, expert systems, management workstations, design	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number): This report describes a system design for a prototype software acquisition manager's workstation. The development of this system will apply software engineering, microcomputer-based personal workstation, and knowledge-based expert system technology in the support of management tasks. The goal of the prototype development is to demonstrate generic characteristics of an application workstation for augmenting the management skills and technical expertise of an acquisition manager.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)



SOFTWARE ACQUISITION MANAGER'S WORKSTATION (SAM/WS)

SYSTEM DESIGN

SAE-DC-84-R-004

April 20, 1984

Software Architecture and Engineering, Inc.
1401 Wilson Boulevard, Suite 1220
Arlington, Virginia 22209

PREFACE

Work on the system-level design for the Software Acquisition Manager's Workstation (SAM/WS) has been supported in part by the Office of Naval Research (ONR) under contract N00014-82C-0428.

Since this document represents a reasonably innovative approach to describing a design, as well as attempting abstract solutions to many complex and poorly understood problems, it is likely that substantial change will occur over a period of time. Any suggestions for improving the approach to specifying a design, particularly for general interactive application systems, or better solutions to particular module design aspects would be welcomed.

TABLE OF CONTENTS

1. INTRODUCTION

This document describes the system design for a Software Acquisition Manager's Workstation (SAM/WS). This design is based on the external requirements definition for the SAM/WS prototype [SAM rqmt].

1.1 OVERVIEW

The SAM/WS is being developed to demonstrate the potential for improving support for software development managers through the application of software engineering technology. While this technology has been used previously in support of designers and programmers, the needs of management have not been addressed. The importance of management decision making in the success of software development, both in terms of cost and product quality, suggests the need for better support.

The primary problem areas of software development management to be addressed are inexperience in the management of software development and lack of technical understanding. The SAM/WS will integrate three technologies that together offer the possibility of reducing these problems: microcomputer-based workstations, knowledge-based expert system technology, and standard management tools. Expert system technology, in particular, will be useful in providing capabilities for assistance in manager decision making. While each of these are generally available separately, no attempt has been made to integrate them into a useful system; the SAM/WS system development will do this.

1.2 SYSTEM DESCRIPTION

The SAM/WS is intended to support the activities of a software acquisition manager. The design views the system as the combination of a generic, hardware/software workstation facility and additional, application-specific software. The generic components provide application-independent capabilities for hardware independence and sophisticated user interface, data storage, and application development components of interactive application systems. The application-specific components of the current design address two acquisition management subactivities: requirements definition and acquisition package development. Software supporting other subactivities or extensions of these will be added to the design incrementally in the future. Within the requirements definition subactivity, the design addresses the determination of required computer and software standards that apply to an acquisition. Based on user-supplied information characterizing the system to be acquired and applicable constraints on the acquisition, the SAM/WS will identify required and suggested standards which apply to the acquisition and guidance for tailoring these standards to the particular acquisition. Within the acquisition package development subactivity, the design addresses support for contract package development for the full scale development portion of the acquisition process. The SAM/WS will provide automatic generation of incomplete acquisition package components (i.e., contract documents), with facilities for completion and tailoring of them to the needs of the particular acquisition. In addition, the SAM/WS will have facilities for tutorial explanation of workstation use and of the acquisition process for inexperienced users.

1.3 GENERAL REFERENCES

[SAM rqmt] Software A & E, Inc. Software Acquisition Manager's Workstation (SAM/WS) - External Requirements Definition, SAE-DC-83-R-021, November 4, 1983.

[SCR design] K. H. Britton, P. C. Clements, A. Parker, D. L. Parnas, J. Shore. A-7E Software Module Guide, Naval Research Laboratory, Washington, D.C. 20375, NRL Memorandum Report 4702, December 8, 1981.

[SCR stdorg] K. H. Britton, D. L. ParnasA Standard Organization for Specifying Abstract Interfaces, Naval Research Laboratory, Washington, D.C. 20375, NRL Report in preparation November 1983.

2. MODULE DECOMPOSITION

A module decomposition defines a conceptual view of the characteristics of a software system design. The decomposition described here presents a design based on the principle of information hiding, modeled on [SCR design]. Following this principle, a module is characterized by the information about some aspect of the system design hidden within the implementation of that module. Such "secrets" are represented to other modules only via an explicitly defined interface that defines the information in an abstract form that is insensitive to potential changes in implementation. The objective of this approach to design is to produce a system which is easy to change in anticipated ways and is easy to understand due to localization of information.

The purpose of this guide is to give the reader interested in some aspect of the system the ability to locate the particular module which implements that aspect. The module decomposition results in a hierarchy of modules such that at each level in the hierarchy, each aspect of the system which is likely to change is the responsibility of exactly one module at that level. Each module at a given level may be further decomposed into a set of modules that together represent the information for which the parent module is responsible. This decomposition proceeds until each terminal module can be further decomposed only if secrets are shared between some of the components. Figure 2.1 depicts the SAM/WS module decomposition as a guide to the following textual description.

HARDWARE HIDING MODULE (HH)
- VIRTUAL COMPUTER MODULE (VC)
- VIRTUAL DEVICES MODULE (VD)
  - VIRTUAL DISPLAY MODULE (CRT)
  - VIRTUAL PRINTER MODULE (PRT)
  - VIRTUAL MASS STORAGE MODULE (STR)

SYSTEM SOFTWARE MODULE (SS)
- DATA FACILITY MODULE (DF)
  - DATA STORAGE MODULE (DST)
  - DATA MODELS MODULE (MOD)
- COMPUTER EXTENSIONS MODULE (CE)
  - ABSTRACT DATA TYPE MODULE (TYP)
  - ABSTRACT LANGUAGE MODULE (LNG)
  - SYSTEM CONFIGURATION MODULE (CFG)
- USER INTERFACE MODULE (UI)
  - VIRTUAL DISPLAY WINDOW MODULE (WIN)
  - INPUT HANDLER MODULE (INP)
  - DISPLAY EDIT/FORMAT MODULE (EDF)
  - EXTERNAL FORMS MODULE (FRM)
- APPLICATION DEFINITION AIDS MODULE (AD)
  - PACKAGE INTEGRATION MODULE (PKI)
  - EXPERT SYSTEM MODULE (EXP)
  - ABSTRACT OBJECT MODULE (OBJ)

APPLICATION SOFTWARE MODULE (AS)
- SAM GENERAL EXPERT MODULE (GE)
  - PROJECT DOMAIN ENTRY/EXIT MODULE (PDA)
  - CONTEXT DEFINITION MODULE (CDF)
  - PRODUCT DEVELOPMENT MODULE (PDV)
  - TUTORIAL ASSISTANCE MODULE (TUT)
  - UTILITY SERVICES MODULE (UTL)
- ACQUISITION REQUIREMENTS DEFINITION MODULE (AR)
  - APPLICABLE POLICIES AND STANDARDS SPECIALIST MODULE (PSS)
- ACQUISITION PACKAGE DEVELOPMENT MODULE (AP)
  - STATEMENT OF WORK SPECIALIST MODULE (SWS)
  - CONTRACT DATA REQUIREMENTS LIST SPECIALIST MODULE (DRS)
  - WORK BREAKDOWN STRUCTURE SPECIALIST MODULE (WBS)
  - SPECIFICATION SPECIALIST MODULE (SPS)
  - REQUEST FOR PROPOSAL SPECIALIST MODULE (RPS)

FIGURE 2.1. SAM/WS MODULE DECOMPOSITION

2.1 LEVEL 1 DECOMPOSITION

At the top level, the SAM/WS system is decomposed into three modules: hardware hiding, system software, and application software. This decomposition was chosen to accommodate a natural view of which module embodies particular information about system function.

The hardware hiding module represents all information about the underlying hardware used to implement the system. Hardware characteristics that are likely to change are abstracted in virtual device descriptions so that changes can be accommodated without changes to either of the other modules. The primary secrets of this module are the actual hardware and software interfaces required by the hardware components of the SAM/WS. Secondary secrets are the data structures and algorithms which implement the virtual devices provided.

The system software module provides software functions and data structures that are of general use in any potential workstation, regardless of application. This module is defined to adequately support the SAM application as currently defined but is more general to allow flexibility, both to accommodate different applications and to support extension of SAM/WS capabilities. The primary secrets of this module are the implementations of its interfaces.

The application software module embodies the requirements of the SAM application as defined in sections 2 and 3 of [SAM rqmt]. Changes in SAM/WS requirements cause changes in the implementation of this module. The primary secrets of this module are the SAM requirements and how user visible effects are determined.

2.2 LEVEL 2 DECOMPOSITION - HARDWARE HIDING MODULE

The hardware hiding module is decomposed into two modules: virtual computer and virtual devices.

The virtual computer module hides characteristics of general purpose computers likely to be used for workstation implementation. The primary secrets of this module are the computer's instruction set, the number of processors, concurrent processing capabilities, and physical memory and architecture characteristics.

The virtual devices module hides characteristics of peripheral devices likely to be used in a workstation implementation. The secrets of this module are the characteristics of these peripheral devices that are likely to change if the physical devices are replaced.

2.3 LEVEL 2 DECOMPOSITION - SYSTEM SOFTWARE MODULE

The system software module is decomposed into four modules: data facility, computer extensions, user interface, and application definition aids.

The data facility module defines structures and functions for logical data storage and access. The secrets of this module are how data is physically stored and retrieved or otherwise derived.

The computer extensions module provides a higher level view and abstracts the logical capabilities of the virtual computer through abstract data type, programming language, and system construction facilities. The secrets of this module are how the necessary data and programs are implemented.

The user interface module provides an extension of hardware hiding module facilities for system interaction with the user. This module defines facilities for sophisticated user input and output, including multiple windows and external formatting of application objects. The secrets of this module are the programs and data structures necessary to provide these facilities.

The application definition aids module provides facilities which are useful in defining conceptual objects and functions for an application. These facilities allow the use of domain-independent expert system technology, integration of separately developed application packages, and access to conceptual models of application objects and associated operations. The secrets of this module are the programs and data structures necessary to provide these facilities.

2.4 LEVEL 2 DECOMPOSITION - APPLICATION SOFTWARE MODULE

The application software module is decomposed into three modules: SAM general expert, acquisition requirements definition, and acquisition package development.

The SAM general expert module implements the functions of the SAM general expert described in section 3.1 of [SAM rqmt]. This module provides all of the user facilities needed to use the workstation in a SAM context. These facilities include helping the user identify products to be developed, understand the operation of the workstation, and better understand software acquisition management and software engineering technology. Facilities of general use in product development are provided. The secrets of this module are the general requirements for supporting SAM activities, including how application specialists are coordinated and share information.

The acquisition requirements definition module defines the products of SAM associated with the acquisition requirements definition phase. The secret of this module is the form and content of these products and their derivation.

The acquisition package development module defines the product of SAM associated with the acquisition package development phase. The secret of this module is the form and content of these products and their derivation.

2.5 LEVEL 3 DECOMPOSITION - VIRTUAL COMPUTER MODULE

The virtual computer module is decomposed into a number of modules. This decomposition will not be described at this time. All facilities will be accessed through system software module facilities.

2.6 LEVEL 3 DECOMPOSITION - VIRTUAL DEVICE MODULE

The virtual device module is decomposed into three modules: virtual display, virtual printer, and virtual mass storage.

The virtual display module defines the characteristics of CRT input/output devices with bit-mapped or character, color or monochrome display and ascii character input keyboard with program defined function keys and user-movable cursor. The secrets of this module are the actual hardware and software interfaces for keyboard input and image display between a CRT and the computer.

The virtual printer module defines the characteristics of a hardcopy output device for ascii character and bit-map graphics output. The secrets of this module are the actual hardware and software interfaces for image output to a printer from the computer.

The virtual mass storage module defines the characteristics of a data storage device based on fixed and removable media which allows logical file definition and direct and sequential access to data pages. The secrets of this module are the actual hardware and software interfaces for storage and retrieval of data on mass storage by the computer and the association between logical and physical storage.

2.7 LEVEL 3 DECOMPOSITION - DATA FACILITY MODULE

The data facility module is decomposed into two modules: data storage and data models.

The data storage module provides facilities for definition of abstract data storage. Access to this abstract storage is provided through various data model interfaces (e.g., relational). The secrets of this module are how abstract storage is constructed in terms of logical storage facilities and how data models determine the placement of data in logical storage.

The data models module provides access to data not physically stored in abstract storage but derivable from other data. Such modelled data is derived through application of filtering and extrapolation functions. The secrets of this module are the formal models of data relationships that define the filtering and extrapolation functions and the implementation of these models.

2.8 LEVEL 3 DECOMPOSITION - COMPUTER EXTENSIONS MODULE

The computer extensions module is decomposed into three modules: abstract data type, abstract language, and system configuration.

The abstract data type module provides facilities for definition and use of abstract data types. Application-specific type derivation is supported. The secrets of this module are the representation of data values and the implementation of operations on each type.

The abstract language module defines concrete programming language interfaces based on an abstract programming language interface to the facilities of the virtual computer. Several languages, including Lisp, Fortran, and C, are supported, each with its own interface definition. The secrets of this module are the implementations of each language.

The system configuration module provides for construction of application modules and of application systems from component modules. Facilities are provided for tailoring of module implementations, selection of alternative implementations of a module, selection of a set of modules for executable system composition, and construction and validation of an application system. The secrets of this module are the representation of application modules and systems and the programs and data structures for their construction and manipulation.

2.9 LEVEL 3 DECOMPOSITION - USER INTERFACE MODULE

The user interface module is decomposed into four modules: virtual display window, input handler, display edit/format, and external forms.

The virtual display window module provides for the definition of virtual windows of variable size and position on the virtual display. Facilities are included for association of internally formatted data with a window for display. The secrets of this module are the representation of virtual windows, the mechanisms for obtaining and displaying data in a window, and the implementation of window operations.

The input handler module provides facilities for processing input data to create logical inputs independent of input mechanism. The secrets of this module are the mechanisms for obtaining and identifying input data and associating it with a display window.

The display edit/format module provides facilities for formatting and modifying displayable objects, particularly text valued objects. The secrets of this module are the internal representation of data objects with formatting guidelines associated and the transformations necessary between internal and external representations to implement the formatting and modification facilities.

The external forms module allows for definition of application-defined forms (templates, frames) in an external representation for use in data display and input. These form definitions can be parameterized for filling and interpreting of fields with variable content. The secrets of this module are the internal representation of these forms and the programs needed to support parameterization and data access.

2.10 LEVEL 3 DECOMPOSITION - APPLICATION DEFINITION AIDS MODULE

The application definition aids module is decomposed into three modules: package integration, expert system, and abstract object.

The package integration module provides for the integration of separately developed packages into an application system. Facilities are provided for defining package interfaces that define the formal parameters of package functions and application object access functions to be used for data access by the package. The secrets of this module are the programs and data structures used to pass data between a package and the rest of a system.

The expert system module provides facilities for the use of domain independent expert system technology in an application system. These include knowledge base definition and access functions that support reasoning and control, explanation, and justification of this reasoning in application object terms. The secrets of this module are the internal representation of knowledge, the implementation of inferencing techniques for reasoning, the mechanisms used to support control, explanation, and justification, and the mechanisms for modifying application object information.

The abstract object module provides for the definition, management, and use of abstract application objects and actions. Types of objects can be defined, instantiated (named), and used as parameters of abstract actions associated with concrete application functions. Objects can be associated with other objects, have explanation text attached, and have data attributes and functional attachments. The secrets of this module are the internal representations of objects, attributes, and attachments.

2.11 LEVEL 3 DECOMPOSITION - SAM GENERAL EXPERT MODULE

The SAM general expert module is decomposed into five modules as defined in section 3.1 of [SAM rqmt]: project domain entry/exit, context definition, product development, tutorial assistance, and utility services. The secrets of each of these modules are the respective functions required.

2.12 LEVEL 3 DECOMPOSITION - ACQUISITION REQUIREMENTS DEFINITION MODULE

The acquisition requirements definition module is decomposed into one module as defined in section 3.2 of [SAM rqmt]: applicable policies and standards specialist. The secrets of this module are the rules and mechanisms for determining standards applicable to an acquisition context.

2.13 LEVEL 3 DECOMPOSITION - ACQUISITION PACKAGE DEVELOPMENT MODULE

The acquisition package development module is decomposed into five modules as defined in section 3.2 of [SAM rqmt]: statement of work specialist, contract data requirements list specialist, work breakdown structure specialist, specification specialist, and request for proposal specialist. The secrets of each of these modules are the rules and mechanisms for producing the associated products.

3. MODULE DEFINITIONS

For each of the level 3 modules identified in the preceding section, the system-level design specifies the design of an interface. An interface is a abstract definition of facilities provided by a module for access to capabilities implemented within that module. A module provides only those facilities that require knowledge of the secrets of that module for implementation. The interface defines what the implementors of client modules can assume will remain static regardless of underlying implementation changes. It also defines what the implementor of the module has to implement (given that unused facilities need not be implemented).

Along with each module's interface, the specification provides justifications for its design, to be used as a guide for implementation and future design revisions. This justification includes assumptions made by the designer that justify what facilities the module should have, a description of issues considered that suggested alternative designs, and guidance to the implementor for approaches that would satisfy the design.

3.1 NOTATION AND STANDARD ORGANIZATION

The organization of the module specifications and the notation used within them is derived from [SCR stdorg]. The notation consists of standard bracketing symbols used as an abbreviation mechanism. Any bracketed identifier is separately defined in a dictionary within the specification so that descriptions using the identifier can be concise and omit redundant information. The bracketing adds information by categorizing all identifiers into a small number of classes as follows:

+ident+ "ident" is the name of a facility of the module that can be referenced at execution time by client modules

++ident++ "ident" is the name of a facility of the module that can be referenced at system creation time by client modules

[ident] "ident" is an abstract data type which can be used as specified as a parameter to the facilities of a module; [XXX ident] can be used to refer to a data type defined in another module (identified by its abbreviated name "XXX")

!ident! "ident" represents some aspect of the abstract internal state of the module that is necessary to adequately characterize the operation of certain facilities

%ident% "ident" is a description of a constraint on the use of a runtime facility that specifies how to avoid incorrect use of that facility

%%ident%% "ident" is a description of a constraint on the use of a system creation facility that specifies how to avoid incorrect use of that facility

Each module specification has an introductory paragraph and two major subsections, an interface definition and design support. The introductory paragraph characterizes the role of the module in the overall system. The interface definition has three components:

exported facilities the facilities available for reference by client modules: each facility has (1) an identifier by which it is referenced, (2) a set of parameters each of which is specified as some abstract data type and some mode of use (I: input, O: output, I/O: input/output, I_opt: input optional, O_opt: output optional, O_ret: output returned), (3) a set of constraints that indicate what constitutes improper use of the facility that could lead to incorrect results, and (4) a description of the results of invoking the facility;

a local dictionary the definition of all bracketed terms used in defining exported facilities;

information hidden a description of the secrets that characterize the module and its facilities.

Design support consists of four components:

interface assumptions

assumptions made by the designer that justify the facilities provided by the module: an assumption indicates why certain facilities are sufficient for expected uses or justify the form facilities take on the basis of external constraints on the implementation; discovery of an invalid assumption usually requires module redesign;

design issues — alternative approaches considered in the design of some aspect of the modules interface: a design issue is some question on the form the interface should take about which several alternatives were considered; the approach taken is justified in terms of its benefits relative to those alternatives;

implementation/configuration information — nonbinding guidance from the designer to the implementor of the module: this includes any ideas or assumptions the designer has about how the module should be implemented or configured for use with other modules; also the designer may anticipate that the module's facilities will be used in limited ways that the implementor should enforce;

references — identification of published papers that influenced the interface design, describe implementations of similar systems, or discuss related concepts.

3.HH.VC Virtual Computer (VC) Module

The virtual computer module defines the components and facilities of an abstract computer that can be represented in software that executes on a general purpose computer system. This module allows the development of a software system that is independent of the instruction set, data types, and physical characteristics of a particular computer system and, thus, reduces the difficulty of moving the software to different hardware.

3.HH.VC.1 Interface Definition

3.HH.VC.1.1 Exported Facilities

Facilities of the VC module are subdivided into four areas: data manipulation, sequence control, concurrency control, and external device access. Facilities in each area are described only in general terms at this time since all will be accessible only via the facilities of the System Software/Computer Extensions/Abstract Language module.

Data Manipulation Functions

(1) provides several primitive type classes and constructors from which all data objects are defined:

type classes: real, integer, timeinterval, bitstring, character, semaphore, reference

contructors: entity, array

(2) provides functions for:

definition of simple data types with (constrained) characteristics of a type class
construction of typed data entities
construction of arrays of typed data
assignment, comparison, and computational operations on typed entities and arrays

Sequence Control Functions

(1) functions for definition of functions with typed parameters and a body consisting of program statements

(2) program statement constructs for parameterized, recursive function invocation, sequential statement execution, repetition of a set of program statements with a mechanism for conditional termination, conditional execution of a set of program statements, and exclusive conditional statement grouping that executes only a single statement set associated with a true condition
(3) constructs for defining, raising, and handling undesired events
(4) functions for creation and use of timers for measuring real time intervals and for signalling completion of time periods

Concurrency Control Functions

(1) functions for definition of static processes that execute an associated function either when specific events occur or at regular intervals
(2) functions for definition, instantiation/invocation, and termination of dynamic processes (within the context of a static process)
(3) identification of regions of program statements to exclude concurrent execution of potentially interferring statements of a set of processes

External Device Access Functions

(1) access for synchronous control and data input/output on ports to external hardware devices
(2) definition of semaphores for the recording of asynchronous data input from external hardware devices

3.HH.VC.1.2 Local Dictionary

3.HH.VC.1.3 Information Hidden

1. The physical components and structure of the computer(s) that are used to implement the virtual computer.

2. The software mechanisms used to implement the functions and constructs of the virtual computer.

3.HH.VC.2 Design Support

3.HH.VC.2.1 Interface Assumptions

3.HH.VC.2.2 Design Issues

3.HH.VC.2.3 Implementation/Configuration Information

1. The facilities assumed to be provided by this module are modelled on Reference 1 from the NRL Software Cost Reduction project. That document provides examples of the form VC facilities might take in a more complete interface specification.

2. None of the facilities of this module will be implemented directly. All will exist conceptually as a minimal semantic base for the abstract semantics of the interface to the System Software/Computer Extensions/Abstract Language (LNG) module. Particular concrete versions of LNG module interfaces may or may not provide all of the facilities described as supported by the VC module.

3.HH.VC.2.4 References

1. D. L. Parnas, K. H. Britton, D. M. Weiss, P. C. Clements, Interface Specifications for the SCR (A-7E) Extended Computer Module, NRL Memorandum Report 4843, Naval Research Laboratory, Washington, D. C., March 29, 1983.

3.HH.CRT Virtual Display (CRT) Module

The virtual display module defines the characteristics of a CRT input/output device consisting of an output display with a user-movable cursor that determines the user's focus of interest and an ascii-mapped input keyboard with additional program-defined function and control keys. A CRT can have either a character or a bitmap display and can produce either color or monochrome images.

3.HH.CRT.1 Interface Definition

3.HH.CRT.1.1 Exported Facilities

Configuration Functions

Name	Parameters	Constraints
++defn_crt_class++	p1:[crt_type];I p2:[displ_type];I p3:[screen_width];I p4:[screen_height];I p5:[color attr];I	

defines the characteristics of a class p1 of CRT devices.

++s_max_crts++ p1:[TYP integer];I
assigns a value to !max CRTs!.

+g_max_crts+ p1:[TYP integer];O_ret
returns the value of !max CRTs!.

Initialization Functions

Name	Parameters	Constraints
+init+	p1:[crt_type];I	%undefnd CRT type%
	p2:[VC device_id];I	%dev slot asgnd%
	p3:[crtid];O_ret	%too many CRTs%

allocates a physical CRT device of type p1 accessible as a physical device named by p2.

+release+ p1:[crtid];I
releases a physical CRT allocation (has no effect if p1 does not represent an allocated CRT).

+g_crt_attr+	p1:[crtid];I	%CRT not defined%
	p2:[displ_type];0	
	p3:[screen_width];0	
	p4:[screen_height];0	
	p5:[color attr];0	

returns the characteristics of CRT p1.

+defn_cursor+	p1:[crtid];I	%CRT not defined%
	p2:[TYP displ_elem];I	%no bitmap capab%
	p3:[offset];I	

defines the visible form p2 of the cursor for (bitmap) CRT p1; the offset p3 (measured relative to the lower left corner of p2) determines a point !focus! of the cursor on the CRT screen at any time.

Input/Output Functions

Name	Parameters	Constraints
+read_keybd+	p1:[crtid];I	%CRT not defined%
	p2:[key];O_ret	

returns the [key] p2 corresponding to the next key (combination) depressed on the keyboard of CRT p1.

+write_image+	p1:[crtid];I	%CRT not defined%
	p2:[TYP displ_elem];I	%no bitmap capab%
	p3:[area];I	

replaces the contents of [area] p3 of the screen of CRT p1 so that image p2 is displayed with the upper left corner of p2 in the upper left corner of p3; the characteristics of p2 (e.g., color, font) will be taken as advice on how to display the image but may vary to satisfy CRT constraints; if a needed characteristic of p2 has not been defined, an arbitrary choice will be made.

Cursor Control Functions

Name	Parameters	Constraints
+s_cursor_posn+	p1:[crtid];I	%CRT not defined%
	p2:[offset];I	%invalid area%

moves the cursor so that its image is displayed with its !focus! at [offset] p2 of the screen of p1.

+g_cursor_posn+	p1:[crtid];I	%CRT not defined%
	p2:[offset];O	

returns the [offset] p2 on the screen of p1 at which the cursor !focus! is currently located.

+enable/disable_cursor+		
	p1:[crtid];I	%CRT not defined%

allows/prevents user movement of the cursor associated with CRT p1 (movement is enabled when the CRT is initialized).

3.HH.CRT.1.2 Local Dictionary

[area]	a [locn], which defines the lower left corner of a rectangular partition of a CRT screen, and an [offset] to the partition's upper right corner, which defines the partition's size.
[bm_screen_height]	a [TYP integer] representing the number of !pixel!s in the vertical dimension of the CRT screen.
[bm_screen_width]	a [TYP integer] representing the number of !pixel!s in the horizontal dimension of the CRT screen.
[ch_screen_height]	a [TYP integer] representing the number of character lines on the CRT screen.
[ch_screen_width]	a [TYP integer] representing the number of character columns on the CRT screen.
[cntl key]	a [TYP char] identifying a user input which can be interpreted as a CRT control action.
[color attr]	enumerated: $color$ or $monochrome$.
[crtid]	a unique identifier for an allocated CRT device.
%CRT not defined%	a [crtid] is being used that does not represent an allocated CRT device.
[crt_type]	a [LNG name] representing a class of physically equivalent CRT devices.
%dev slot asgnd%	the indicated [VC device_id] is already in use for some other device.
[displ_type]	[TYP enum : $char$ or $bitmap$].
!focus!	the [locn] defining the position of the cursor within !screen area!.
[func key]	a [TYP integer] identifying a key or key combination having no CRT defined meaning.
%invalid area%	an [area] is referenced which is not contained completely within !screen area! for a [crtid].
[key]	the [TYP union] of ([TYP char], [func key], [cntl key]).

[locn]	an [offset] from the lower left corner of a CRT screen defining a !pixel! on the screen.
!max CRTs!	the maximum number of CRTs that can be used concurrently in the system.
%no bitmap capab%	the specified CRT cannot display an image whose definition includes bitmaps.
[offset]	a list of [TYP integer]s, the first of which represents a horizontal length and the second of which represents a vertical length on a CRT screen; these lengths are specified in the same units as [screen height] and [screen width].
!pixel!	the smallest unit on a CRT screen that can be displayed.
!screen area!	an [area] with [locn] equal to (0,0) and [offset] determined by +g_crt_attr+.
[screen height]	[ch_screen_height] or [bm_screen_height] depending on an associated [displ_type].
[screen width]	[ch_screen_width] or [bm_screen_width] depending on an associated [displ_type].
%too many CRTs%	!max CRTs! are currently allocated.
%undefnd CRT type%	no CRT class has been defined with name pl.

3.HH.CRT.1.3 Information Hidden

1. The hardware and software interfaces to physical display devices.

3.HH.CRT.2 Design Support

3.HH.CRT.2.1 Interface Assumptions

1. This module can be configured to support several types of physical CRT input/output device. Each type can be distinguished as either for character or for bitmap display and as for either color or monochrome image display.

2. Every CRT will be associated one-to-one with a Virtual Computer device id. The device id determines the actual physical routing of I/O. Each CRT must be allocated exactly once before use and must be deallocated afterwards to allow reuse of the Virtual Computer device id.

3. The form of the cursor displayed on a bitmap CRT screen can be modified to be any bitmap image. The form and focus point of the cursor on a character screen is fixed.

4. It is possible to detect the depressing of a key on the CRT keyboard. Each key (and some combinations of keys) can be mapped into either the Virtual Computer character set ([TYP char]) or represents CRT control or program-definable function input. Undefined keys or key combinations either are not detected or have an unpredictable effect.

5. It is possible to modify the contents of a CRT screen to display a specified image in a specified area of the screen. This is restricted in that an image created from a bitmap cannot be displayed on a character screen CRT.

6. The position of the cursor on a CRT screen can be determined or modified.

3.HH.CRT.2.2 Design Issues

1. Should this interface provide for use of conventional as well as bitmap CRT devices? It is desirable to provide limited workstation facilities on conventional CRTs. Much of the functionality of intended workstation applications are text oriented and can be presented on such CRTs.

2. What level of graphics capabilities should this interface assume? While some graphical display is useful (e.g., for partitioning the screen into windows or for icon menus), this is within the bounds of normal bitmap display. Some, such as window boundaries, are also possible with character display. More complex graphics CRTs are not likely to be used as a workstation-controlling device. A workstation to support an application needing such capabilities is beyond the scope of this design.

3.HH.CRT.2.3 Implementation/Configuration Information: None.

3.HH.CRT.2.4 References: None.

3.HH.PRT Virtual Printer (PRT) Module

The virtual printer module defines the characteristics of hardcopy output devices intended for ascii character (with variable font) or bit-map graphics output.

3.HH.PRT.1 Interface Definition

3.HH.PRT.1.1 Exported Facilities

Name	Parameters	Constraints
++defn_prt_class++	p1:[prt_type];I p2:[displ_type];I p3:[page_width];I p4:[page_length];I	

defines the characteristics of a class p1 of hardcopy printers.

+init+	p1:[prt_type];I	%undefnd PRT type%
	p2:[VC device_id];I	%dev slot asgnd%
	p3:[prtid];O_ret	

allocates a physical hardcopy printer of type p1 accessible as a physical device named by p1.

+release+	p1:[prtid];I	

releases a physical printer allocation (has no effect if p1 does not represent an allocated printer).

+g_prt_attr+	p1:[prtid];I	%PRT not defined%
	p2:[displ_type];O	
	p3:[page_width];O	
	p4:[page_length];O	

returns the characteristics of printer p1.

+write_image+	p1:[prtid];I	%PRT not defined%
	p2:[TYP displ_elem];I	%no bitmap capab%

provides for output of image p2 on printer p1.

3.HH.PRT.1.2 Local Dictionary

[bm_page_height] a [TYP integer] representing the number of !pixel!s in the vertical dimension of the printer page.

[bm_page_width] a [TYP integer] representing the number of !pixel!s in the horizontal dimension of the printer page.

[ch_page_height]	a [TYP integer] representing the number of character lines on the printer page.
[ch_page_width]	a [TYP integer] representing the number of character columns on the printer page.
%dev slot asgnd%	the indicated [VC device_id] is already in use for some other device.
[displ_type]	[TYP enum : $char$ or $bitmap$].
%no bitmap capab%	the specified printer cannot display images created from bitmaps.
[page_height]	[ch_page_height] or [bm_page_height] depending on an associated [displ_type].
[page_width]	[ch_page_width] or [bm_page_width] depending on an associated [displ_type].
%PRT not defined%	a [prtid] is being used that does not represent an allocated printer.
[prtid]	a unique identifier for an allocated printer.
[prt_type]	a [LNG name] representing a class of physically equivalent printers.
%undefnd PRT type%	no printer class has been defined with the given name.

3.HH.PRT.1.3 Information Hidden

1. The hardware and software interfaces to physical hardcopy printers.

3.HH.PRT.2 Design Support

3.HH.PRT.2.1 Interface Assumptions

1. This module can be configured to support several types of physical hardcopy print devices. Each type can be distinguished as either for character or for bitmap display. All produce a monochrome image.

2. Every printer will be associated one-to-one with a Virtual Computer device ID. The device ID determines the actual physical routing of output. Each printer must be allocated exactly once before use and must be deallocated afterwards to allow reuse of the Virtual Computer device ID.

3. Depending on device type (a character type printer cannot receive a bitmap image), it is possible to cause a hardcopy image of a bitmap or text string to be generated on the output media.

3.HH.PRT.2.2 Design Issues: None.

3.HH.PRT.2.3 Implementation/Configuration Information: None

3.HH.PRT.2.4 References: None.

3.HH.STR Virtual Mass Storage (STR) Module

The virtual mass storage module defines the characteristics of devices for persistent data storage. Both fixed and removal storage components are available for use.

3.HH.STR.1 Interface Definition

3.HH.STR.1.1 Exported Facilities

Name	Parameters	Constraints
+define_file+	p1:[file_id];I p2:[TYP type];I p3:[TYP type];I p4:[access_key];0_ret	

provides for definition of a logical data storage file p1 containing entries of type p2; each value in the domain of type p3 uniquely selects one of the entries of p1; p4 provides a key for owner control of file access.

+g/s_access+	p1:[file_id];I p2:[access_key];I p3:[access];I p4:[access_key];0_ret	

provides an access key p4 with access rights defined by p3 to file p1 where p2 is the file owner's access key.

+read+	p1:[file_id];I p2:[access_key];I p3:[entry_id];I p4:!entry!;0 p5:[TYP boolean];0_ret	

provides for retrieval of an entry p4 identified by p3 in file p1; p5 = $FALSE$ indicates that no entry existed for p3 so that the read failed.

+lock+	p1:[file_id];I p2:[access_key];I p3:[entry_id];I p4:[TYP boolean];I p5:[write_lock];0 p6:[TYP boolean];0_ret	

reserves the entry identified by p3 in file p1 for use with write lock p5; p4 = $TRUE$ indicates that the call waits for write permission to continue while p4 = $FALSE$ indicates no wait if the entry is in use; p6 = $FALSE$ (when p4 = $FALSE$) indicates that the entry was in use and could not be locked.

+write+ p1:[write_lock];I
p2:!entry!;I
causes replacement of the file entry locked with write lock p1 by value p2.

+delete+ p1:[write_lock];I
causes deletion of the file entry locked with write lock p1.

+unlock+ p1:[write_lock];I
returns the write lock p1 allowing the reserved file entry to be released for subsequent write access.

3.HH.STR.1.2 Local Dictionary

[access]	[TYP enum : $read$, $write$, $control$].
[access_key]	a unique identifier that gives particular access rights to a particular file.
[entry_id]	a value in the domain of the index for a file.
[file_id]	a [VC name] uniquely representing a file.
[write_lock]	a unique identifier which provides write/delete access control of a particular entry of a file.

3.HH.STR.1.3 Information Hidden

1. .

3.HH.STR.2 Design Support

3.HH.STR.2.1 Interface Assumptions

1. Data storage can be viewed as a set of logical files consisting of typed entries, each of which is distinguished by the domain values of an index type. Access to each file can be controlled by a unique access key generated when the file is created. Restricted access rights can be provided by generation of new access keys given a known access key.

2. Access to files are needed to read, write, and delete entries of a file. Write access requires the ability to lockout concurrent access to a record.

3.HH.STR.2.2 Design Issues

1. How to map file entries into physical storage (e.g., hashing of the index value, sequential as created, sequential on index value).

3.HH.STR.2.3 Implementation/Configuration Information

3.HH.STR.2.4 References: None.

3.DF.DST Data Storage (DST) Module

The data storage module provides facilities for the definition, storage, and access of persistent data. Three models of data store structure are supported: relational, network, and data space. These can be used independently or in combination to most conveniently provide data storage.

3.DF.DST.1 Interface Definition

3.DF.DST.1.1 Exported Facilities

Relational Data Storage

Name	Parameters	Constraints
++RDataBase++	p1:[name];I p2:[owner_key];O_ret	

creates a relational database for permanent data storage; an owner key p2 is created for controlling access.

++relation++	p1:[database_name];I p2:[name];I p3::TYP list: of [attribute];I p4:[candidate_key];I p5::TYP list: of [candidate_key];I	

creates a (null valued) relation named by p2 in database p1 consisting of attributes p3 of which attributes identified by p4 is a primary key and p5 identifies a set of alternate keys.

++virtual_reln++	p1:[database_name];I p2:[name];I p3:[reln1_expr];I	

defines a relation named by p2 logically, but not physically, a member of database p1 that is equivalent to relational expression p3; all [relation]s referenced in p3 must be real or virtual members of p1.

+acquire_[database_name]+	p1:[RDB];O_ret	

initiates access to the specified database.

+release+	p1:[RDB];I	

terminates access to the database associated with p1.

+s_[relation_name]+	p1:[RDB];I p2:[relation];I	

assigns relation p2 as the value of the specified relation in database p1.

+g_[relation_name]+ p1:[RDB];I
p2:[relation];0_ret
returns p2, the specified relation in database p1.

+union+ p1:[relation];I
p2:[relation];I
p3:[relation];0_ret
returns a relation p3 which is the !union! of relations p1 and p2.

+diff+ p1:[relation];I
p2:[relation];I
p3:[relation];0_ret
returns a relation p3 which is the !difference! of relation p1 from relation p2.

+product+ p1:[relation];I
p2:[relation];I
p3:[relation];0_ret
returns a relation p3 which is the !product! of relations p1 and p2.

++selector_op++ p1:[type];I
p2:!TYP list! of [selector_def];I
defines operators for use in !theta selection! of relations on attributes of type p1.

+select+ p1:[relation];I
p2:[selector];I
p3:[relation];0_ret
returns a relation p3 which is the !theta selection! p2 of relation p1.

+project+ p1:[relation];I
p2:[set] of [attribute_name];I
p3:[relation];0_ret
returns a relation p3 which is the !projection! p2 of relation p1.

Network Data Storage

Name	Parameters	Constraints

++NDataBase++ p1:[name];I
p2:[owner_key];0_ret
creates a network database named by p1 for permanent data storage; an owner key p2 is created for controlling access.

++virtual_NDB++ p1:[name];I
p2:[database_name];I
defines a virtual network database named by p1 contained in network database p2 (real or virtual).

```
++record++              p1:[real_database_name];I
                        p2:[name];I
                        p3:!TYP list! of [attribute];I
```
defines a class of record named by p2 for database p1 consisting of the attributes p3.

```
++virtual_record++      p1:[virtual_database_name];I
                        p2:[name];I
                        p3:!TYP list! of ([name], [record_name], [attribute
                        name]);I
```
defines a record named by p2 in virtual database p1 which is a composite of attributes from the network database containing p1.

```
++set++                 p1:[database_name];I
                        p2:[name];I
                        p3:[owner_spec];I_opt
                        p4:!TYP list! of [member_spec];I
```
defines a class p2 of !set! for database p1 with owner records characterized by p3 and member records characterized by p4; if p3 is not input, a singular set is specified which has no explicit owner.

```
+open_[database_name]+ p1:[currency];O_ret
```
initiates access to the specified database with a currency p1 context.

```
+close+                 p1:[currency];I
```
terminates access to the database associated with the currency p1 context.

```
+find_[record_name]+    p1:[currency];I
                        p2:[currency];O_ret
```
returns a currency p2 which reflects changes to currency p1 necessary to make a (new) record of the type indicated by "[record_name]" accessible.

```
+find_[record_name]_in_[set_name]+
                        p1:[currency];I
                        p2:[currency];O_ret
```
returns a currency p2 which reflects changes to currency p1 necessary to make a (new) record of the type indicated by "[record_name]" in the !set! associated with the current owner of set type "[set_name]" accessible.

```
+find_[set_name]_owner+
                        p1:[currency];I
                        p2:[currency];O_ret
```
returns a currency p2 which reflects changes to currency p1 necessary to make the owner of the set of type "[set_name]" in which the current record is a member accessible.

```
+find_[set_name]_member+
                        p1:[currency];I
                        p2:[currency];0_ret
   returns a currency p2 which reflects changes to currency p1 necessary to
   make a member of the set of type "[set_name]" of which the current record
   is the owner accessible.

+get_[record_name]+     p1:[currency];I
                        p2:[record];0_ret
   returns the current record p2 of type indicated by "[record_name]" in the
   currency p1 database.

+store_[record_name]+   p1:[currency];I
                        p2:[record];I
                        p3:[currency];0_ret
   stores record p2 of type indicated by "[record_name]" in the currency p1
   database so that currency p3 results.

+erase_[record_name]+   p1:[currency];I
                        p2:[currency];0_ret
   removes the current record of type indicated by "[record_name]" from the
   currency p1 database so that currency p2 results.

+erase_[set_name]_members+
                        p1:[currency];I
                        p2:[currency]0_ret
   removes all records of the database in the currency p1 context which are
   members of a set of set type indicated by "[set_name]" whose owner is the
   current record in currency p1 so that currency p2 results.

+modify_[record_name]+  p1:[currency];I
                        p2:[record];I
                        p3:[currency];0_ret
   replaces the current record of type indicated by "[record_name]" in the
   currency p1 database with record p2 so that currency p3 results.

+connect_[record_name]_to_[set_name]+
                        p1:[currency];I
                        p2:[currency];o_ret
   adds the current "[record_name]" type record to the current "[set_name]"
   type set in the currency p1 database so that currency p2 results.

+disconnect_[record_name]_from_[set_name]+
                        p1:[currency];I
                        p2:[currency];0_ret
   removes the current "[record_nam   ype record from the current "[set
   name]" type set in the currency p1 database so that currency p2 results.
```

Data Space Storage

Name	Parameters	Constraints
++DataSpace++	p1:[name];I p2:[owner_key];O_ret	

creates a data space named by p1.

++Super_DSpc++	p1:[name];I p2::TYP list: of [DSpc_name];I	

defines a data space named by p1 which is a superset consisting of data spaces p2.

++entity_class++	p1:[DSpc_name];I p2:[TYP type];I p3:[name];I	

defines an entity class named by p3 in data space p1 whose member entities are of type p2.

++entity_ref_class++	p1:[DSpc_name];I p2:[DSpc_name];I p3:[ent_cl_name];I p4:[name];I	

defines an entity class named by p4 in data space p1 that can be used to reference the value of entity class p3 in data space p2.

+environ+	p1:[DSpc_name];I p2:[name];I	

creates a referencing environment named by p2 in data space p1 such that every entity name defined in p1 refers to exactly one entity in p2.

+replicate+	p1:[environ_name];I p2:[name];I	

creates a referencing environment named by p2 in the same data space as environment p1 with all entities of p2 being copied from p1.

+acquire+	p1:[environ_name];I p2:[context];O_ret	

establishes an exclusive access context p2 to data space environment p1.

+release+	p1:[context];I	

releases access context p1.

+g_[ent_cl_name]+	p1:[context];I p2:[value];O_ret	

returns the value p2 of the "[ent_cl_name]" entity in the context p1 data space; if the entity named is an entity reference, the value of the referenced entity is returned.

+s_[ent_cl_name]+ p1:[context];I
p2:[value];I

assigns p2 as the value of the "[ent_cl_name]" entity in the context p1 data space.

+ref_[ent_cl_name]+ p1:[context];I
p2:[environ_name];I

establishes the "[ent_cl_name]" reference entity in the context p1 data space such that the appropriate entity class in environment p2 (which must be in the appropriate data space) is referenced.

3.DF.DST.1.2 Local Dictionary

[attribute]	a !TYP list! specifying an [attribute_name] and a [TYP type].
[attribute_name]	a [name] that uniquely identifies an attribute within a set of attributes that comprise a relation.
[attribute_value]	a [LNG value] of the type associated with a particular [attribute].
[database_name]	a [name] which uniquely identifies a database.
[name]	a [LNG name].
[owner_key]	a unique identifier that gives access control of a database to its creator.

Relational terms

[candidate_key]	a !TYP list! of [attribute_name] (nonempty) which can be used to uniquely identify a tuple in the relation containing the attributes named; none of the named attributes can be removed from the relation without endangering this uniqueness property.
!difference!	given two [relation]s composed of the same [attribute]s, all [tuple]s that occur in a designated one of the [relation]s with all [tuple]s that occur in the other omitted.

!product!	given two [relation]s having no [attribute_name]s in common, a [relation] consisting of a [tuple] for each pair of [tuple]s from those [relation]s, where that [tuple] includes an [attribute] for each [attribute] of each [relation].
!projection!	a [relation] consisting of all [tuple]s of another [relation] with only specified [attribute]s of that [relation] included.
[relation]	a [TYP set] of [tuple]s, all of which are defined by the same set of [attribute]s.
[relation_name]	a [name] which uniquely identifies a real or virtual relation of a database.
[reln1_expr]	a [relation_name] or a [reln1_func] or a !TYP list! containing two elements: a !TYP list! of [attribute]s and a [relation].
[reln1_func]	a [LNG expr] consisting entirely of relational operations to produce a [relation] type output.
[selector]	a !TYP list! specifying an [attribute_name], a [selector_id] defined for the same type as this attribute in the relation to which !theta selection! is to be applied, and a value (possibly another [attribute name]), also of the same type.
[selector_def]	a !TYP list! specifying a [selector_id] and an equivalent [TYP boolean]-valued [LNG func_id] that accepts two input parameters of a specified [TYP type].
[selector_id]	a [name] which uniquely identifies selectors for a given [type].
!theta selection!	a [relation] consisting of all [tuple]s of another [relation] that satisfy a constraint on the value of one of its [attribute]s as defined by a specified [selector].
[tuple]	a [TYP 1bl_setun] of [attribute_value]s associated with the [attribute]s that define the containing [relation].
!union!	given two [relation]s composed of the same [attribute]s, all [tuple]s that occur in either one of the [relation]s, without repetition of any duplicates.

Network terms

[currency]	information that provides a context for access to a network database; identifies a particular database and, within it, a current record, a current record of each defined record type, and a current record of each defined set.
[insert_order]	[TYP enum : $first$, $last$, $next$, $prior$, key, any].
[member_spec]	a list specifying the [record_name] that characterizes set members and a [set_key] for this type of member.
[owner_spec]	a [record_name] that characterizes set owners.
[record]	a [TYP lbl_setun] of [attribute_value]s associated with the [attribute]s which defined a particular database record type.
[record_name]	a [name] which uniquely identifies a type of database record.
!set!	an association between one record, distinguished as the set "owner", and a collection of other records, characterized as set "members".
[set_key]	a !TYP list! of two elements: a [TYP set] of [attribute_name]s (whose values can be used to uniquely identify a set member) and an [insert_order].
[set_name]	a [name] which uniquely identifies a database !set!.

Data Space terms

[context]	an identifier which provides access to a data space referencing environment.
[DSpc_name]	a [name] associated with a data space definition.
[ent_cl_name]	a [name] associated with an entity class or entity reference class definition of a data space.
[environ_name]	a [name] associated with a data space referencing environment definition.
[value]	a [LNG value] of a type associated with an entity class definition within a data space.

3.DF.DST.1.3 Information Hidden

1. How data stores are represented and stored.

2. How data store entries are created, positioned, and subsequently located.

3. The implementation of operations on relations and sets.

3.DF.DST.2 Design Support

3.DF.DST.2.1 Interface Assumptions

1. A data store is a grouping of logically related data. The conceptual organization and elements of a data store are static while the actual contents are dynamic but persistent (values can change but are retained until an element is discarded).

2. Three models of data store structure, access, and element characteristics are useful. These are relational, network, and data space.

3. The relational model views a database as a collection of "relations" which are unordered collections of homogeneous "tuples". Every relation is in third normal form (see Chapter 9 of Reference 2). A tuple is an unordered collection of typed data items. Each tuple in a relation contains a single value for each data item (or the item may be undefined). Relations can be stored into or retrieved from a database and can be input or output of five types of relational algebra operations: union, difference, extended cartesian product, selection, and projection. All other useful operations can be composed from these.

3.DF.DST.2.2 Design Issues

1. Three models of data storage are supported by this module: relational, network, and data space (or heap). These seem to be the major extremes currently in use for management of data storage resources. The network model provides a file-oriented approach to storage and access of persistent data. The data space model provides an approach oriented to dynamic allocation of free space independent of logical associations among data values. The relational model provides an intermediate approach that groups data into "relations" that represent functional dependencies among data items grouped together but is independent of logical associations among these relations.

2. How to support the use of any of the data store models with data defined using one of the other models? Or should there be a single definition facility set with several access models?

3. How to support locking/exclusion in concurrent data access? (resource control facility in LNG?)

4. How to allow implicit data space environment access associated with a user process? (e.g., in T-Lisp which executes a program body within the scope of a "locale" construct of data items)

3.DF.DST.2.3 Implementation/Configuration Information

1. The definition of the relational interface is derived from the abstract relational model described in Reference 1. The interface varies as follows: (1) relational operations apply to unnamed as well as named relations (the validity of this requires verification); (2) due to (1), the product operation can be applied only to relations that have no attribute names in common (ambiguity would result, however this deviation is not desirable); (3) the theta selection operation is not restricted to ordering relationships (per se) between values of a type but can be based on any valid comparison between two values of a type that delivers a boolean result (this would allow decision for each data type about how to treat undefined (null) values); (4) virtual relations are considered the domain of the MOD module and are omitted here.

2. The definition of the network interface is derived from the descriptions in Part 4 of Reference 2. Not all capabilities of the DBTG network model are provided explicitly or in the same form, particularly implicit operations in that model.

3.DF.DST.2.4 References

1. Date, C. J., "A Formal Definition of the Relational Model", ACM SIGMOD Record, 13, 1, September 1982, 18-29.

2. Date, C. J., An Introduction to Database Systems, Addison-Wesley Publishing Company, 1977.

3.DF.MOD Data Models (MOD) Module

The data models module provides abstract models for the definition of data not physically stored but derivable from other data.

3.DF.MOD.1 Interface Definition

3.DF.MOD.1.1 Exported Facilities

3.DF.MOD.1.2 Local Dictionary

3.DF.MOD.1.3 Information Hidden

1. .

3.DF.MOD.2 Design Support

3.DF.MOD.2.1 Interface Assumptions (to be defined)

3.DF.MOD.2.2 Design Issues: None.

3.DF.MOD.2.3 Implementation/Configuration Information: None.

3.DF.MOD.2.4 References: None.

3.CE.TYP Abstract Data Type (TYP) Module

The abstract data type module provides abstract definitions of data representations and operations on those representations. Each representation can have several implementations, each appropriate to particular usage needs. Data definitions are categorized into two general type classes: scalar valued and collection valued.

3.CE.TYP.1 Interface Definition

3.CE.TYP.1.1 Exported Facilities

Scalar Type Classes

The following scalar type classes are provided: numeric, enumerated, image, character, and union. Basic scalar types are defined as instances of these type classes. Types [boolean], [real], and [integer] are builtin instances of type classes enumerated, numeric, and numeric respectively. The function definitions immediately following are valid for all scalar type data; following these are function definitions unique to each of the five base type classes.

Name	Parameters	Constraints
+eq/neq+	p1:[type];I p2:[type];I p3:[boolean];O_ret	

indicates whether data values p1 and p2 (which must be of the same base type) have equal/nonequal values.

+extrep+	p1:[type];I p2:[charstr];O	

produces a character string p2 which is a human-readable representation of the value of p1.

+intrep+	p1:[charstr];I p2:[type];O_opt p3:[boolean];O_ret	

provides a correctly typed value p2 corresponding to character string p1 if p3 = $TRUE$, indicating a valid value was derivable from p1.

enumerated type class

literal values: a series of characters bracketed by "$"; each type declaration defines the set of strings (i.e., symbolic values) that are applicable to entities of that type.

++enum_type++ p1:[name];I
p2:!list! of [!enum litval!];I
p3:[boolean];I
defines an enumerated type named by p1 consisting of symbolic values in p2, where p3 indicates whether the value set is ordered allowing ordering comparisons.

builtin types: [boolean].

+not+ p1:[boolean];I
p2:[boolean];O_ret
returns the logical complement p2 of boolean p1.

+and+ p1:[boolean];I
p2:[boolean];I
p3:[boolean];O_ret
returns the logical "and" p3 of booleans p1 and p2.

+or+ p1:[boolean];I
p2:[boolean];I
p3:[boolean];O_ret
returns the logical "or" p3 of booleans p1 and p2.

+xor+ p1:[boolean];I
p2:[boolean];I
p3:[boolean];O_ret
returns the logical "exclusive or" p3 of booleans p1 and p2.

image_type class

An "image" is a two-dimensional combination of other images where the unit elements are a "background" image and a "foreground" image.

literal values: none

+"combin_rule"+ p1:[image];I_opt
p2:[image];I_opt
p3:[image];O_ret
produces the image p3 which is the combination of source images p1 and p2 (referred to as S1 and S2 respectively below) under the specified combination rule; p1 and/or p2, respectively, may be omitted only if the combination rule does not refer to S1 and/or S2. Image combination consists of determining whether each unit element of p3 should be "background" or "foreground". A combination rule evaluates corresponding unit elements of images p1 and p2 to determine a truth value where "false" is equivalent to "background" and "true" is equivalent to "foreground" for the corresponding unit element in the image p3. The combination rules are: "BkGnd", "FrGnd", "S1F_and_S2F", "S1F_and_S2B", "S1F", "S1B_and_S2F", "S2F", "S1F_xor_S2F", "S1F_or_S2F", "S1B_and_S2B", "S1B_xor_S2F", "S2B", "S1F_or_S2B", "S1B", "S1B_or_S2F", "S1B_or_S2B".

```
+clip+                  p1:[image];I
                        p2:[offset];I
                        p3:[extent];I
                        p4:[image];O_ret
```
produces an image p4 with extent p3 derived as a subimage of p1 with an !image origin! at offset p2 from the !image origin! of p1.

```
+extend+                p1:[image];I
                        p2:[offset];I
                        p3:[extent];I
                        p4:[image];O_ret
```
produces an image p3 which contains the image p1 with its !image origin! at offset p2 from the !image origin! of p1; any area of p3 not filled by p1 will be filled with background image.

```
+g_extent+              p1:[image];I
                        p2:[extent];O_ret
```
returns the extent p2 of image p1.

numeric type class

builtin types: [real] which has no unit of measurement and [integer] which is a subtype of [real] with a resolution of one

literal values: standard decimal notation (e.g., 123.22, .0034, 256) or exponent notation (i.e., [real]E[integer] which represents [real] * 10 ** [integer]; e.g., 2.7E3 which is equivalent to 2700.) followed where appropriate by a units identifier in parentheses (e.g., 35(mph)).

universal constraints: %out of range%

```
++num_type++            p1:[name];I
                        p2:!list! of [units];I
```
defines a numeric type p1 where p2 identifies valid units of measurement of values of this type.

```
++interval++            p1:[name];I
                        p2:[!num type!];I
                        p3:[range];I
                        p4:[resol];I_opt
```
defines a subtype p1 of numeric type p2 restricted to range p3 with a resolution p4 (1 if omitted); the elements of the range specification must be a precise multiple of p4.

```
+real_to_[units]+       p1:[real];I                          %units in error%
                        p2:[!interval type!];O_ret
```
returns a value of the type of p2 of quantity equal to p1 when p2 is measured in the specified units.

```
+[units]_to_real+       p1:['interval type'];I              %units in error%
                        p2:[real];0_ret
```
returns a real value p2 which measures the quantity of p1 in the indicated units.

```
+leq/lt/geq/gt+         p1:[numeric];I                      %incompat opnds%
                        p2:[numeric];I
                        p3:[boolean];0_ret
```
determines whether the value of p1 is less than or equal/less than/greater than or equal/greater than the value of p2.

```
+[type]_min/max+        p1:[numeric];0_ret
```
returns the minimum/maximum value in the domain of the indicated numeric type.

```
+incr/decr+             p1:[numeric];I
                        p2:[numeric];0_ret
```
returns the minimum/maximum value p2 in the domain of the type of p1 which is greater/less than p1.

```
+add+                   p1:[numeric];I                      %incompat opnds%
                        p2:[numeric];I
                        p3:[numeric];0_ret
```
returns the sum p3 of p1 and p2; all operands must be the same numeric type.

```
+sub+                   p1:[numeric];I                      %incompat opnds%
                        p2:[numeric];I
                        p3:[numeric];0_ret
```
returns the result p3 of subtracting p2 from p1; all operands must be the same numeric type.

```
++mult_opnds++          p1:[type];I
                        p2:'list' of [units];I
                        p3:[type];I
                        p4:'list' of [units];I_opt
                        p5:[type];I
                        p6:'list' of [units];I
```
defines the subtypes that are valid as parameters of the multiplication operation: p5 defines the type of the result where p1 and p3 (interchangably) define the types of the input operands; p2, p4, and p6 (which must have the same number of elements) define the input and result units for the operation (p4 may be omitted if p3 is of type [real]); multiplication is valid by default for unitless numerics.

```
+mult+                  p1:[numeric];I                      %incompat opnds%
                        p2:[numeric];I
                        p3:[numeric];0_ret
```
returns the product p3 of p1 and p2.

```
++div_opnds++          p1:[type];I
                       p2:!list! of [units];I
                       p3:[type];I
                       p4:!list! of [units];I_opt
                       p5:[type];I
                       p6:!list! of [units];I
```
defines the subtypes that are valid as parameters of the division operation: p5 defines the type of the result where p1 and p3 define the types of the input operands; p2, p4, and p6 (which must have the same number of elements) define the input and result units for the operation (p4 may be omitted if p3 is of type [real]); division is valid by default for unitless numerics.

```
+div+                  p1:[numeric];I                    %incompat opnds%
                       p2:[numeric];I
                       p3:[numeric];0_ret
```
returns the quotient p3 of dividing p1 by p2.

```
+mod+                  p1:[numeric];I                    %incompat opnds%
                       p2:[numeric];I
                       p3:[numeric];0_ret
```
returns the modulo p3 of p1 relative to p2.

```
+absv+                 p1:[numeric];I
                       p2:[numeric];0_ret
```
returns the absolute value p2 of p1.

```
+comple+               p1:[numeric];I
                       p2:[numeric];0_ret
```
returns the numeric complement p2 of p1.

```
+truncate+             p1:[numeric];I
                       p2:[numeric];0_ret
```
returns the maximum value p2 in the domain of the type of p1 which has an integer magnitude less than that of p1.

```
+round+                p1:[numeric];I
                       p2:[numeric];0_ret
```
returns the value p2 in the domain of the type of p1 which is the integer magnitude closest in value to that of p1.

Character type class

literal values: any element of the ASCII character set.

Union Type Class

The union type class allows type definitions in which the domain of values is a discriminated union of the set of values of a set of member types. Each member type is distinguished by a label for use in access.

++union_type++ p1:[name];I
p2:!list! of !memb_descr!;I
defines a union type p1 whose values are one of the fields identified in p2.

+!memb name!+ p1:[union];I
p2:[boolean];0_ret
determines whether the value of union p1 is the named field's definition.

+g_!memb name!+ p1:[union];I
p2:[!memb type!];0_ret
returns the value p2 of the named field in union p1 (the result is unspecified if the union has the value of a different field).

+s_!memb name!+ p1:[!memb type!];I
p2:[union];0_ret
returns the union p2 with the value of p1 corresponding to the named field.

Collection Type Classes

Name	Parameters	Constraints

Sequenced Multiset Type Class

A "sequenced multiset" is an implicitly ordered collection of elements, all of the same type, such that any value in the domain of the type can occur zero or more times in the collection.

literal values: a !typed list! of [slot_val]s.

++seq_type++ p1:[name];I
p2:[slot_type];I
defines a sequence type named by p1 with value members of type p2.

+!seq_type!+ p1:[seq];0_ret
creates an empty sequence p1 of the indicated sequence type.

+empty+ p1:[seq];I
p2:[boolean];0_ret
determines whether sequence p1 contains any elements.

+g_first/last+ p1:[seq];I
p2:[seq];0_opt
p3:[slot_val];0_ret
returns the value p3 of the first/last slot of sequence p1; optionally outputs the sequence p2 which is identical to p1 with the first/last value slot omitted.

```
+remove_first/last+      p1:[seq];I
                         p2:[slot_val];0_opt
                         p3:[seq];0_ret
```
returns the sequence p3 identical to sequence p1 with the first/last value slot omitted; optionally outputs the value p2 of the first/last slot of p1.

```
+add_first/last+         p1:[seq];I
                         p2:[slot_val];I
                         p3:[seq];0_ret
```
creates a sequence p3 identical to p1 with value p2 added as a new first/last entry.

Set Type Class

A "set" is a collection of elements, all of the same type, such that every value in the domain of the type is in the collection exactly zero or one time.

```
++set_type++             p1:[name];I
                         p2:[slot_type];I
```
defines a set type named by p1 with value members of type p2.

```
+!set_type!+             p1:[set];0_ret
```
creates an empty set p1 of the indicated set type.

```
+empty+                  p1:[set];I
                         p2:[boolean];0_ret
```
determines whether set p1 is empty.

```
+insert/remove+          p1:[set];I
                         p2:[slot_val];I
                         p3:[set];0_ret
```
creates a set p3 identical to set p1 with value p2 added/removed (a removal has no effect if p2 is not a member of p1).

```
+member+                 p1:[set];I
                         p2:[slot_val];I
                         p3:[boolean];0_ret
```
indicates whether value p2 is a member of set p1.

```
+extract+                p1:[set];I
                         p2:[set];0
                         p3:[slot_val];0_ret
```
creates a set p2 identical to set p1 with an arbitrary member value p3 removed.

Indexed Multiset Type Class

An "indexed multiset" is a collection of elements, all of the same type, with an associated "index" such that any value in the domain of the type has zero or more associated values from the domain of the type of the index by which the value can be referenced (i.e., the indexed multiset defines a one-to-many mapping from the index domain to the value domain).

```
++idx_mset_type++      p1:[name];I
                       p2:[type];I
                       p3:[index_type];I
```
defines an indexed multiset type p1 with value members of type p2, each of which is uniquely identified by a value from the domain of index type p3.

```
+!imset_type!+         p1:[imset];O_ret
```
defines an indexed multiset p1 with no member values.

```
+member+               p1:[imset];I
                       p2:[index_val];I
                       p3:[boolean];O_ret
```
indicates whether indexed multiset p1 contains a member value for index value p2.

```
+s_elem+               p1:[imset];I
                       p2:[index_val];I
                       p3:[imset_val];I
                       p4:[imset];O_ret
```
returns a indexed multiset p4 identical to indexed multiset p1 with the member value identified by index value p2 set to value p3.

```
+g_elem+               p1:[imset];I
                       p2:[index_val];I
                       p3:[imset_val];O_ret
```
returns the member value p3 of indexed multiset p1 identified by index value p2.

Labelled SetUnion Type Class

A labelled setunion is a collection of elements, each of which has an associated name, such that each element has a specified type and may or may not have a defined value as a member of the collection.

```
++lbl_setunion_type++  p1:[name];I
                       p2:!list! of !memb descr!;I
```
defines a labelled setunion type p1 consisting of at most one value member for each !memb descr! in p2.

```
+!lset_type!+          p1:[lbl_setun];O_ret
```
defines a labelled setunion p1 which has no value members.

```
+has_!memb name!+        p1:[lbl_setun];I
                         p2:[boolean];O_ret
```
indicates whether the labelled setunion p1 contains a value for the specified member name.

```
+s_!memb name!+          p1:[lbl_setun];I
                         p2:[!memb type!];I
                         p3:[lbl_setun];O_ret
```
defines a labelled setunion p3 identical to labelled setunion p1 with the specified member value set to p2.

```
+g_!memb name!+          p1:[lbl_setun];I
                         p2:[!memb type!];O_ret
```
returns the value p2 of labelled setunion p1 indicated by the specified member name.

Derived Types

Derived types are types that are of general usefulness for the class of systems being designed but not considered inherently primitive.

Character string type class

```
literal values:          a contiguous sequence of one or more [char] (e.g., B,
                         256, (.), XmA) enclosed in double quotes ("ABC").

++charstr_type++         p1:[name];I
                         p2:[integer];I
```
defines a character string type p1 which has a maximum length p2.

```
+null+                   p1:[charstr];O_ret
```
returns a zero length charstr p1.

```
+replc+                  p1:[charstr];I
                         p2:[range];I_opt
                         p3:[charstr];I_opt
                         p4:[charstr];O_ret
```
returns the string p4 as a copy of string p1 with substring indicated by p2 replaced by string p3; if p2 is not input, p3 is appended to the end of p1; if p3 is not input, string p4 is string p1 with the substring indicated by p2 removed (replaced by a zero length string).

```
+substr+                 p1:[charstr];I
                         p2:[range];I
                         p3:[charstr];O_ret
```
returns the string p3 which is the substring of p1 indicated by range p2.

```
+len+                    p1:[charstr];I
                         p2:[integer];O_ret
```
returns an integer p2 which indicates the length of character string p1.

Display Medium

Name	Parameters	Constraints

+medium+ p1:[displ_medium];0_ret

creates a display medium p1 that can be used to define display attributes for external presentation of data.

+g/s_font+ p1:[displ_medium];I
p2:[font];0_ret/I

returns/defines the character font p2 to be used in bitmap displaying of text characterized by display medium p1.

+g/s_color+ p1:[displ_medium];I
p2:[color];0_ret/I

returns/defines the color p2 in which data characterized by display medium p1 are to be displayed.

+g/s_bkgnd_color+ p1:[displ_medium];I
p2:[color];0_ret/I

returns/defines the color p2 of the background on which data characterized by display medium p1 are to be displayed.

+invert_color+ p1:[displ_medium];I
p2:[boolean];I

indicates whether the color and background color of data characterized by display medium p1 should be reversed relative to its context.

+g/s_underline+ p1:[displ_medium];I
p2:[boolean];0_ret/I

indicates/defines whether text defined with medium p1 is to be underlined.

+g/s_highlight+ p1:[displ_medium];I
p2:[boolean];0_ret/I

indicates/defines whether data characterized by medium p1 is to be highlighted.

+g/s_blink+ p1:[displ_medium];I
p2:[boolean];0_ret/I

indicates/defines whether data characterized by medium p1 is to be blinked (blinking is the same as turning highlighting on and off periodically).

+merge+ p1:[displ_medium];I
p2:[displ_medium];I
p3:[displ_medium];0_ret

returns medium p3 which results from merging the features of mediums p1 and p2 so that features of p2 override those of p1.

+complete+ p1:[displ_medium];I
p2:[boolean];0_ret

indicates whether all features of p1 are defined.

Display Objects

+text+ p1:[displ_medium];I
p2:[charstr];I
p3:[displ_elem];O_ret
creates a display element p3 corresponding to the character string p2 with display attribute changes defined by medium p1.

+graphic+ p1:[displ_medium];I
p2:[image];I
p3:[displ_elem];O_ret
creates a display element p3 corresponding to the image p2 with display attribute changes defined by medium p1.

+g_charstr+ p1:[displ_elem];I
p2:[charstr];O_ret
provides a character string p2 represented by display element p1.

+g_image+ p1:[displ_elem];I
p2:[image];O_ret
provides an image p2 represented by display element p1.

+g_medium+ p1:[displ_elem];I
p2:[displ_medium];O_ret
returns the medium p2 which defines the display attribute changes applicable to element p1.

+append+ p1:[displ_obj];I_opt
p2:[displ_elem];I
p3;[displ_obj];O_ret
creates display object p3 as object p1 with element p2 appended.

+join+ p1:[displ_obj];I
p2:[displ_obj];I
p3:[displ_obj];O_ret
creates display object p3 as the fusion of objects p1 and p2.

+g_next_elem+ p1:[displ_obj];I
p2:[displ_elem];O_ret
returns the next unaccessed display element p2 in object p1.

+reset+ p1:[displ_obj];I
makes all display elements of object p1 appear unaccessed.

Fonts

++font++ p1:!list! of (([char], [image]) pairs);I
p2:[font];O_ret
creates a character font p2 made up of the character/image associations defined by p1.

```
+g_text+                p1:[font];I
                        p2:[charstr];I
                        p3:[image];0_ret
   returns an image p3 which represents character string p2 in font p1.
```

Colors

```
+g_color+               p1:[base color];I
                        p2:[color shade];I
                        p3:[color];0_ret
   creates a color p3 corresponding to shade p2 of base color p1.

+g_base_color+          p1:[color];I
                        p2:[base color];0_ret
   identifies the base color p2 of color p1.

+g_color_shade+         p1:[color];I
                        p2:[color shade];0_ret
   identifies the shade p2 of color p1.
```

3.CE.TYP.1.2 Local Dictionary

[base color]	[enum : red, $green$, $blue$, $black$, $white$, $grey$, $orange$, $purple$, $brown$, $yellow$].
[boolean]	[enum : $true$, $false$].
[char]	a [type] in the character type class.
[charstr]	a derived [type] for representing character strings (i.e., sequences of [char] values).
[color]	a derived [type] that represents a visible color.
[displ_elem]	a derived [type] used to represent a displayable value.
[displ_medium]	a derived [type] used to represent display characteristics of a [displ_elem].
[displ_obj]	a derived [type] used to represent a composite of [displ_elem]s with associated [displ_medium]s.
[color_shade]	[enum : $light$, $medium$, $dark$].
[enum]	a [type] in the enumerated type class

[extent]	a [lbl_setun] of (horiz:[integer],vert:[integer]) corresponding to an [offset] that represents the size of an [image] in unit [image]s.
[font]	a derived [type] that specifies a mapping between [char] type values and [image] type values.
[image]	a [type] in the image type class.
[imset]	a [type] in the indexed multiset type class.
!imset_type!	the [name] of an [imset].
[imset_val]	a value of the [type] associated with a particular [imset].
%incompat opnds%	numeric operands must be of the same type.
[index_type]	an [enum], an !interval type! [numeric] with a finite domain, or a [lbl_setun] all of whose members are of type [index_type].
[index_val]	a value of the [type] associated as an index with a particular [imset].
[integer]	a [numeric] subtype having a resolution of 1.
!interval type!	the [name] of a [numeric] subtype that has a [range] with a finite minimum or maximum value.
[lbl_setun]	a [type] in the labelled setunion type class.
!list!	a series of elements bracketed by parentheses and separated by commas (e.g., "(1,2,3)" or "(AB, "XYZ")").
!memb descr!	a !typed name!.
!memb name!	the [name] part of a !memb descr!.
!memb type!	the [type] part of a !memb descr!.
[name]	an [LNG name].
[numeric]	a [type] in the numeric type class including [real], [integer], defined numeric types (those with associated units of measurement), and derived !interval type!s.

[offset]	a !list! of two [integer]s, the first of which represents a horizontal number of unit [image]s and the second of which represents a vertical number of unit [image]s.
%out of range%	the result of a numeric operation is out of the [range] specified as valid for the result.
[range]	a !list! of two [real]s, the first of which defines a minimum value and the second of which defines a maximum value; the literal "INF" can be used in either position to represent an indeterminate minimum or maximum value.
[real]	a [numeric] subtype which has no associated units of measurement.
[resol]	a positive-valued [real] which indicates the minimum resolution at which numeric values of a given type can be distinguished.
[seq]	a [type] in the sequenced multiset type class.
[set]	a [type] in the set type class.
[slot_type]	the [type] of an element in a [seq] or a [set].
[slot_val]	the value of an element in a [seq] or a [set].
[type]	a data type defined in or using the facilities of this module; a [name] used in defining a data type.
!typed name!	a [name] followed by a colon (":") followed by a [type] which indicates the type associated with use of the name.
!typed list!	a !list! followed by a colon (":") followed by a [type] which indicates the type of all elements of the !list!.
[union]	a [type] in the union type class.
[units]	a [name] which represents a unit of measurement of a [numeric].
%units in error%	an incorrect "units" identifier is used in reference to a specified !interval type!.

3.CE.TYP.1.3 Information Hidden

1. The representation of values within each of the data types.

2. The implementation of operations associated with a data type.

3.CE.TYP.2 Design Support

3.CE.TYP.2.1 Interface Assumptions

1. All scalar data can be characterized as either numeric, enumerated, image, or character valued. All more complex data can be characterized as a collection of values composed from values in these four scalar classes. Data may also be characterized as having a value from the union of the domains of two or more classes of data.

2. All data collections can be characterized as a set, a sequenced multiset, an indexed multiset, or a labelled setunion of some type of data (either scalar or collection).

3.CE.TYP.2.2 Design Issues

1. Initially, storage allocation was included as a facility of this module. It was concluded that this was not a proper concern and was independent of data type specifications. The goal of this module is to provide definitions of abstract type specifications while other modules can better determine how to allocate physical storage to hold entities with typed values. Considering storage allocation here leads to confusion, particularly in considering dynamic allocation and the issues of short-term versus long-term retention.

2. Some languages (e.g., Ada) provide generic type specifications (using discriminants) that allow parameterized data types that are instantiated as several specific types later. (An example is a generic "square" parameterized by an integer that represents the length of its sides; specific "square" types of fixed size can then be defined as instances of the generic type.) Such facilities need not be provided by this module in that translation of the language can, using a generic type specification, transform a subsequent instantiation into one of the specific type definitions of this module.

3.CE.TYP.2.3 Implementation/Configuration Information

1. Any abstract type referenced in a program written in a concrete programming language must be implemented either in that same language or in the language in which it is implemented. This may lead to several implementations of each data type and will require care to maintain consistency between these implementations. It may be useful to support more control by each module client as to what characteristics are needed (e.g., save space, fast insertion, fast searching) and may lead to categories of data type implementation (e.g., array versus list implementation of the sequence type).

2. Related to the preceding issue is the issue of whether the facilities of this module should be purely functional (no hidden side effects) or have internal storage, particularly for implementation of compound types such as sets and sequences. This should be transparent to a client of these facilities, but it may be desirable to allow client control in some abstract way. The preferred implementation is probably as macros in the source language of each client program.

3.CE.TYP.2.4 References: None.

3.CE.LNG Abstract Language (LNG) Module

The abstract language module defines facilities of programming languages for describing computations that can be evaluated by the virtual computer. This specification describes the facilities of these languages in a generic form as an informal guide to the semantics of a concrete language that supports a given facility. Concrete specifications are provided separately for any languages represented by this module. Any particular concrete language may provide only a subset of the described facilities and restrict the computational descriptions that are possible. Concrete languages anticipated include, but are not restricted to, Lisp, C, and Ada.

3.CE.LNG.1 Interface Definition

3.CE.LNG.1.1 Exported Facilities

Data Manipulation

Name	Parameters	Constraints
++entity++	p1:[name];I p2:[TYP type];I p3:[const value];I_opt p4:$Const$;I_opt	

defines an entity named by p1 of type p2 with initial value p3 (undefined if p3 is not input); if p3 and p4 are input, p4 indicates that p1 identifies a fixed value entity.

+entity+	p1:[TYP type];I p2:[value];I_opt p3:[entref];O_ret	

creates a reference p3 to an entity of type p1 with initial value p2 (undefined if p2 is not input).

+undefine+	p1:[entity];I	

causes data item p1 to have an undefined value relative to its type domain.

+undefined+	p1:[entity];I p2:[boolean];O_ret	

determines whether data item p1 has an undefined value relative to its type domain.

+set+ p1:[entity];I
p2:[!base type!];I
p3:[!base type!];O_ret

causes data item p1 to be assigned value p2, where p1 is the same type as p2; returns the value of p2 assigned to p1.

+swap+ p1:[entity];I
p2:[!base type!];I
p3:[!base type!];O_ret

causes data item p1 to be assigned value p2, where p1 is the same type as p2; returns the value of p1 before the assignment.

(In addition to these functions, concrete language definitions will provide definitions of concrete data types for the implementation of the Abstract Data Type module.)

Sequence Control

Name	Parameters	Constraints

++program++ p1:[name];I
p2:!seq! of [param];I
p3:!seq! of [name];I
p4:!list! of (!version name!, !prog impl! pairs);I

defines a program named by p1 which has parameters identified by p2 and exception programs identified by p3.

++prog_impl++ p1:[prog_name];I
p2:[name];I
p3:[statement];I

defines statement p3 to be an implementation version named by p2 of program p1.

+[prog name].[version name]+
p1:!seq! of [param_value];I
p2:!seq! of [prog name];I

a [statement] which causes the execution of the version "[version name]" of the program identified by "[prog name]" with parameter values p1; p2 identifies programs associated with the set of exception conditions that the invoked program detects.

+[excp name]+

a [statement] which causes the execution of a program associated with the "[excp name]" for the program containing this statement.

+seq+ p1:!seq! of [statement];I

a [statement] which causes the sequence of statements p1 to execute in order.

+cond+ p1:[guard_defn];I_opt
p2:!seq! of [guarded_stmt];I

a [statement] which defines a sequence of guarded statements p2 such that execution of this statement causes the first true guarded statement to be executed; p1 defines guards that are referenced within p2.

+loop+ p1:[statement];I

a [statement] which defines a repetition context for the statement p1.

+loop_cntl+ p1:[statement];I
p2:[loop_cntl];I

a [statement] which causes the execution of statement p1 to set !loop cntl! as indicated by p2 for the containing loop statement.

+skip+

a [statement] which indicates "no action".

Concurrency Control

Name	Parameters	Constraints

Static Processes

++P_Process++ p1:[prog name];I
p2:!seq! of [parameter];I_opt
p3:[period];I
p4:[pprocess_sw];I
p5:[priority];I

defines a periodic process that executes program p3 with parameter sequence p1 with a periodicity of p2 at priority level p5 whenever process switch p4 is on.

++D_Process++ p1:[prog name];I
p2:!seq! of [parameter];I_opt
p3:[event_id];I
p4:[priority];I

defines a demand process that executes program p3 with parameter sequence p1 at priority level p4 whenever event p2 occurs.

Dynamic Processes

+co_stmt+ p1:!set! of [statement];I

a [statement] which causes concurrent activation as dynamic processes of the set of statements p1.

```
+co_expr+              p1:[prog name];I
                       p2:[TYP seq] of [param_value];I
                       p3:[TYP seq] of [value];0
```
a [statement] which applies program p1 concurrently to each of the parameter sets of p2 producing results p3.

```
+fail+
```
a [statement] which cancels the containing dynamic process, so that no output is produced.

```
+succeed+              p1:[value];0_ret
```
a [statement] which cancels the containing dynamic process and returns output p1.

Exclusion Regions

Exclusion regions provide a mechanism for preventing concurrent processes from interfering with each other by executing conflicting statements concurrently.

```
++Region++             p1:[name];I
                       p2:[statement];I
```
defines statement p2 to be a region named by p1.

```
++Exclusion++          p1:!set! of (!list! of ([region_name], [region_name]));I
```
defines a set p1 of asymmetric exclusion relations between pairs of regions, such that execution of the second region of a pair cannot begin while the first is being executed.

Semaphores

Semaphores provide a mechanism for the synchronization of concurrent processes.

```
++Semaphore++          p1:[name];I
                       p2:[integer];I
                       p3:[semaphore];0_ret
```
defines a semaphore p3 named by p1 which has an initial value of p2.

```
+up+                   p1:[semaphore];I
```
increments the semaphore p1.

```
+down+                 p1:[semaphore];I
```
decrements the semaphore p1.

```
+pass+                 p1:[semaphore];I
```
delays the caller while semaphore p1 has a negative value.

3.CE.LNG.1.2 Local Dictionary

[const_value]	a [litval] or a fixed value [entity].
[entity]	a [name] or [entref] which uniquely identifies a typed entity.
[entref]	a unique identifier for a dynamically defined typed entity.
[expression]	
[guard]	a [TYP boolean] valued [expression].
[guard_defn]	(to be defined)
[guarded_stmt]	[guard] "_" [statement] specifies that the [statement] can be executed if and only if the associated [guard] is true.
[litval]	a literal value of a form defined for a data type in the Abstract Data Type module.
[loop_cntl]	[enum : $term$, $cont$].
!loop_cntl!	an indicator for each loop statement that specifies whether the statement should be terminated or repeated at the completion of its current execution instance; this is undefined at the start of each execution instance and must be defined through the execution of a loop control statement within the loop.
[name]	a sequence of printable characters, the first of which must be alphabetic and which includes no spaces.
[param]	
[param_value]	
!seq!	a !TYP list! of elements which is viewed as ordered.
!set!	a !TYP list! of elements which is viewed as unordered.
[value]	[entity] or [litval]

3.CE.LNG.1.3 Information Hidden

1. .

3.CE.LNG.2 Design Support

3.CE.LNG.2.1 Interface Assumptions

1. .

3.CE.LNG.2.2 Design Issues

1. .

3.CE.LNG.2.3 Implementation/Configuration Information: None.

3.CE.LNG.2.4 References

1. D. L. Parnas. An Alternative Control Construct and Its Formal Definition, IBM Technical Report TR FSD-81-0012.

2. D. L. Parnas, K. H. Britton, D. M. Weiss, P. C. Clements, Interface Specifications for the SCR (A-7E) Extended Computer Module, NRL Memorandum Report 4843, Naval Research Laboratory, Washington, D. C., March 29, 1983.

3.CE.CFG System Configuration (CFG) Module

The system configuration module provides facilities for the construction of executable systems.

3.CE.CFG.1 Interface Definition

3.CE.CFG.1.1 Exported Facilities

Name	Parameters	Constraints

3.CE.CFG.1.2 Local Dictionary

3.CE.CFG.1.3 Information Hidden

1. .

3.CE.CFG.2 Design Support

3.CE.CFG.2.1 Interface Assumptions

1. .

3.CE.CFG.2.2 Design Issues

1. .

3.CE.CFG.2.3 Implementation/Configuration Information: None.

3.CE.CFG.2.4 References

1. B. W. Lampson, E. E. Schmidt. "Organizing Software in a Distributed Environment" in Proceedings of the SIGPLAN '83 Symposium on Programming Language Issues in Software Systems (ACM SIGPLAN Notices 18(6)) June 1983, 1-13.

3.UI.WIN Virtual Display Window (WIN) Module

The virtual display window module provides for the definition and use of display "windows" for the concurrent presentation of data object information on a CRT screen. A window is a rectangular space which presents a (partial) view of a data object's external form to a user when the window is visible on a CRT screen.

3.UI.WIN.1 Interface Definition

3.UI.WIN.1.1 Exported Facilities

Initialization Functions

Name	Parameters	Constraints
+displ_obj_map+	p1:[CRT crtid];I p2:[FRM displ_id];I p3:[frm_win_id];0_ret	%unknown CRT%
defines a window p3 for presentation of display form p2 on CRT p1.		
+displ_doc_map+	p1:[CRT crtid];I p2:[EDF source_id];I p3:[edf_win_id];0_ret	%unknown CRT%
defines a window p3 for presentation of source document p2 on CRT p1.		
+g_CRT+	p1:[win_id];I p2:[CRT crtid];0_ret	%undefined window%
returns the CRT p2 with which window p1 is associated.		
+g_displ_id+	p1:[frm_win_id];I p2:[FRM displ_id];0_ret	%undefined window% %not a form window%
returns the identifier p2 for the display form in window p1.		
+g_source_id+	p1:[edf_win_id];I p2:[EDF source_id];0_ret	%undefined window% %not a doc window%
returns the identifier p2 for the source document in window p1.		
+break+	p1:[win_id];I	%undefined window%
deletes a window definition, preventing further reference.		

Window Movement Functions

Name	Parameters	Constraints
+g/s_locn+	p1:[win_id];I p2:[CRT offset];O_ret/I_opt	%undefined window%

returns/sets the upper left corner of the window p1 to coincide with the CRT location p2 if input or the location of the CRT cursor otherwise.

+g/s_size+	p1:[win_id];I p2:[CRT offset];O_ret/I	%undefined window%

returns/sets the size p2 of window p1 measured from its lower left corner.

+expand+	p1:[CRT crtid];I	%unknown CRT%

causes the size of the window within whose visible boundaries the cursor for CRT p1 is positioned to increase in the direction of the edge(s) nearest the current CRT cursor position.

+shrink+	p1:[CRT crtid];I	%unknown CRT%

causes the size of the window within whose visible boundaries the cursor for CRT p1 is positioned to increase in the direction of the edge(s) nearest the current CRT cursor position.

+display+	p1:[win_id];I	%undefined window%

makes window p1 completely visible on its associated CRT screen, possibly by covering previously visible portions of other windows (window size and position on the CRT screen will be assigned arbitrarily if not previously defined); the associated CRT cursor is moved to the upper left corner of p1.

+uncover+	p1:[CRT crtid];I	%unknown CRT%

makes completely visible the window within whose visible boundaries the cursor for CRT p1 is positioned, possibly by covering previously visible portions of other windows (window size and position on the CRT screen will be as last defined); the position of the CRT cursor relative to the window will not change.

Input/Output Functions

Name	Parameters	Constraints
+g_focus+	p1:[CRT crtid];I p2:[win_id];O_opt p3:[CRT offset];O_opt p4:[TYP boolean];O_ret	%unknown CRT%

when p4 = $TRUE$, indicating the cursor for CRT p1 is within the boundaries of some window, p2 identifies the window in whose visible boundaries the cursor is positioned and p3 gives the offset of the cursor focus relative to the upper left corner of the image mapped into that window.

+await_focus_chg+	p1:[CRT crtid];I	%unknown CRT%

delays the caller until the next occurrence of the cursor for CRT p1 being moved across a window boundary.

+scroll+ p1:[CRT crtid];I %unknown CRT%

causes the window containing the CRT cursor to move over its contents a distance determined by the window's width in the direction of the edge(s) nearest the CRT cursor (limited such that the cursor does not move relative to the window contents and remains visible in the window); if none of the window's contents are hidden in the direction indicated, the window image does not change.

3.UI.WIN.1.2 Local Dictionary

[edf_win_id]	a unique identifier for an active window mapped to present the image of an EDF module defined document.
[frm_win_id]	a unique identifier for an active window mapped to present the image of an FRM module defined display form.
%undefined window%	the specified [win_id] does not correspond to a currently defined window.
%unknown CRT%	a specified CRT id does not correspond to a currently active CRT.
[win_id]	either an [frm_win_id] or an [edf_win_id].

3.UI.WIN.1.3 Information Hidden

1. The spatial correspondence between virtual windows and the screen area of an associated physical CRT. The data structures used to represent the relationships between virtual windows associated with a single CRT screen.

2. The relationship between an internal source image and the visible image within the boundaries of a window at a given time. The mechanisms for modifying the portion of an image that is visible due to user-instigated, window-relative changes in focus.

3. The mechanisms for maintaining a window image as a valid reflection of the current state of internal data in a timely manner.

3.UI.WIN.2 Design Support

3.UI.WIN.2.1 Interface Assumptions

1. Every CRT screen is associated with some active user. Windows are a mechanism for logically structuring information displayed to a user so that several items can be viewed independently. Each window is defined as a (partial, movable) view of internal data, formatted into a displayable image.

2. While a window is defined and has nonzero size, it displays a portion of an image to which it has access. A window may be partially or completely hidden by other windows on the CRT screen. This module can determine which windows are visible at any time on the associated CRT screen and can make any invisible or partially hidden window visible, possibly by covering other visible windows. The relationship between image and window guarantees that a "local focus" associated with the image is always within the window.

3. The position and size of a window on the CRT screen can be changed. Positional relationships between windows are recognized so that a window which is overlapped by others will be visible only where not overlapped.

4. The contents of a window are created as a displayable image from internal data by another module which provides an access function for obtaining a current image. If source data for an image is extensive, only a portion of the image will be created, such that its relation to the whole can be determined and other portions accessed as needed. If a window is not large enough to display a complete image, different areas of the image can be viewed by scrolling the available window area over the image.

5. Given the absolute position of the user cursor on the CRT screen, it is possible to determine a single window in which the cursor is positioned and a relative position with respect to the image in that window. This user cursor determines the "global focus" of the user. It is possible to monitor the cursor position so that movement across a window boundary can be reported at the time of occurrence.

3.UI.WIN.2.2 Design Issues

1. What functions are appropriate for window repositioning relative to a display source image or relative to a CRT screen? What parameters are appropriate for each such function? In most cases, it seems awkward to have to specify explicit quantitative measures in modifying the position, size, or visible contents of a window. This is particularly true since it is desirable to support use of both bitmap and character CRT screens. It is useful to provide functions that allow the caller to either base such window requests on the current cursor position where appropriate or simply to indicate the general result desired (e.g., that the window should be made larger or smaller). In the latter case, the effect on the window image should be significant but small enough that a choice between too much and too little will not generally be necessary.

2. How should the need for window image updates be determined? by the window module or by another module that can monitor when changes to the source object occur? It was decided that the window module should be responsible for deciding when to update the contents of a window. This avoids having to reveal to other modules exactly what images are contained in each window. Other modules (e.g., FRM, EDF) may be required to provide an access function that can indicate that a source value has changed to minimize unnecessary window updates.

3.UI.WIN.2.3 Implementation/Configuration Information

1. This module provides for automatic CRT screen updating as internal data images associated with a window change. Display images are obtained from the modules indicated in the mapping functions provided for window creation.

2. As described in design issue 1, window scrolling and size modification can be requested without specifying particular measures. Such scrolling should cause a significant portion, but not all, of the visible image to change. Window expansion or shrinking should cause a window to become some proportion of its current size (say, 10 percent more or less of the CRT screen area in the desired dimension). In both cases, the goal should be to significantly modify the user's view of the window's contents while maintaining the basic focus (in no case should the position of the cursor relative to a source image change as a result of a window repositioning operation). Major changes in focus within a source object are handled by the module that creates the image for window display.

3.UI.WIN.2.4 References: None.

3.UI.INP Input Handler (INP) Module

The input handler module defines virtual keyboards made up of logical keys that can be associated with the context of a window defined for a CRT. Such keyboard/window connections allow contextual interpretation and processing of user inputs.

3.UI.INP.1 Interface Definition

3.UI.INP.1.1 Exported Facilities

Name	Parameters	Constraints
++keybd++	p1:[keybd];O_ret	

defines a logical keyboard p1 for which [key]s can be defined and recognized on input.

++key++	p1:[keybd];I	%%duplicate key%%
	p2:[key_id];I	%%invalid pattern%%
	p3:[key_pattern];I	
	p4:[TYP boolean];I_opt	

defines a logical key p2 on keyboard p1 that is equivalent to a key pattern p3; if p4 = $TRUE$, the case for $ALPHA$ keys is significant.

++drop_char++	p1:[keybd];I
	p2:[key];I

identifies a previously defined key p2 whose input on keyboard p1 makes the preceding character added to !input stream! inaccessible; preceding unaccessed keys remain accessible.

++drop_line++	p1:[keybd];I
	p2:[key];I

identifies a previously defined key p2 whose input on keyboard p1 makes the preceding !input line! in !input stream! inaccessible; preceding unaccessed !input line!s remain accessible.

++line_term++	p1:[keybd];I
	p2:[key];I

identifies a previously defined key p2 to be a !line term! for keyboard p1.

+connect+	p1:[keybd];I
	p2:[WIN win_id];I
	p3:[connection];O_ret

creates an input connection p3 between keyboard p1 and window p2 (which determines an active CRT); if user input occurs while the user cursor is in no window or in a window with which no keyboard is associated, that input is rejected as invalid.

+g_input_stream+ p1:[connection];I
p2:[TYP charstr];O

returns the !input stream! p2 (with [CRT func_key]s removed) for connection p1.

+await_[key]+ p1:[connection];I
p2:[TYP boolean];I

delays the caller until the indicated [key] is at the beginning of the !input stream! for connection p1; p2 indicates whether the [key] is removed from the start of !input stream! allowing processing to continue or remains there until removed by a call to +g_line+; [CRT func_key]s are always removed regardless of p2.

+g_input+ p1:[connection];I
p2:[TYP charstr];O_ret

removes and returns the first !input line! p2 in the !input stream! for connection p1.

+g_input_focus+ p1:[connection];I
p2:[CRT offset];O_ret

identifies the user's focus relative to the upper left corner of the portion of the image mapped into the window of connection p1.

+queue_input+ p1:[connection];I
p2:[TYP boolean];I

if p2 = $TRUE$, interrupts !input stream! processing for connection p1; if p2 = $FALSE$, resumes the processing of !input stream! in sequence.

+dump_input+ p1:[connection];I

empties !input stream! of its current contents for connection p1 without further processing.

+hit_key+ p1:[connection];I
p2:[key];I

inserts key p2 at the end of the !input stream! for connection p1.

3.UI.INP.1.2 Local Dictionary

[compos_key]	[TYP enum : ANY, $ALPHA$, NUM, $SPCHAR$, $FKEY$, $CKEY$] where $ALPHA$ includes all upper and lower case alphabetics and space, NUM is 0 to 9, $SPCHAR$, $FKEY$, and $CKEY$ are, respectively, all special characters, function keys, and control keys defined on a CRT keyboard, and ANY is any [keybd] key.
[connection]	an association between a logical keyboard definition and a window defined for an active CRT that determines how user inputs from that CRT are processed when the user cursor is within the boundaries of that window.

[crt_key] a [TYP charstr] representing a [CRT key] as follows:
alphanumerics: the standard symbol (e.g., "A", "t", "5")
special characters: the standard symbol (e.g., "@", "=") with the exception of "(", ")", "*", "+", "$", and "'" which must be preceded by "'" (e.g., "'*")
function keys: the name of the key bracketed by "$" (e.g., "$F2$")
control keys: the name of the key bracketed by "$" (e.g., "$CD$")

%%duplicate key%% (1) a [key_id] has been defined more than once; or (2) two or more [key_id]s have been defined for a logical keyboard that map into the same sequence of [CRT key]s.

!input line! a [TYP seq] of [CRT key] (omitting [CRT func_key]s and !line term!s) bracketed by !line term!s.

!input stream! [TYP seq] of [CRT key] corresponding to the input received for a [connection] and which has not been accessed; determines the order in which associated inputs are processed.

%%invalid pattern%%

[key] [TYP union] of ([key_id], [crt_key], [compos_key]).

[key_id] a [TYP name] initiated and terminated with "$", excluding the symbolic values of [crt_key] and [symb_key].

[key_pattern] one of:
[key]
[key][key_pattern]
([key_pattern]+[key_pattern]+ ... +[key_pattern])
*[key_pattern]
*[integer],[integer][key_pattern]
(embedded spaces are significant)

[keybd] a unique identifier for the description of a logical keyboard from which input can be received.

!line term! an input key that marks the end of an !input line!; the start of !input stream! is equivalent to a !line term!; when no such key is defined for a keyboard, the end of !input stream! serves as a !line term!.

3.UI.INP.1.3 Information Hidden

1. The mechanisms for detecting CRT keyboard inputs and mapping them into logical keys defined as a context sensitive pattern.

2. The mechanisms and representation for storing and reporting of inputs associated with an input connection.

3.UI.INP.2 Design Support

3.UI.INP.2.1 Interface Assumptions

1. Acceptable input is defined by logical keyboards composed of input keys whose input can be detected in some context. Some keys have meaning in the context of input handling (i.e., backspace, line delete, and end of line) and are not detectable outside of input handling. All other keys are externally detectable in some way. Any input not representing a key on an appropriate logical keyboard is considered an error to be reported to the source CRT.

2. Any CRT keyboard definition can be mapped into any logical keyboard definition; however some keys on the logical keyboard may be inaccessible to the CRT user if the CRT keyboard lacks a full keyset.

3. The interpretation of user inputs depends on a window with which those inputs are associated. Such an association is indicated by the window in which the user cursor is positioned when those inputs occur and the definition of a logical keyboard associated with that window.

4. It is possible to define a window for use in displaying input errors. An error need be visible only until subsequent input is received.

3.UI.INP.2.2 Design Issues

1. How to recognize truncated inputs that are sufficiently long to be distinguished from other possible inputs?

2. The echoing of input, being an output function, is not the responsibility of this module. Since a function is provided for access to the (unprocessed) !input stream! of each connection, another module can map this data into a window for display.

3.UI.INP.2.3 Implementation/Configuration Information

1. Reference 1 describes a conceptual model for a tool for the flexible definition of logical input primitives as the composition of other input primitives (in the context of a complete definition of an "input-output tool"). This influenced the design of this module's interface such that this module could be used to implement such a tool.

3.UI.INP.2.4 References

1. J. van den Bos, M. J. Plasmeijer, P. H. Hartel. "Input-Output Tools: A Language Facility for Interactive and Real-Time Systems", IEEE Transactions on Software Engineering, 9(3), May 1983, 247-259.

3.UI.EDF Display Edit/Format (EDF) Module

The display edit/format module provides facilities for modifying text source data and for formatting of this data for external presentation to a user.

3.UI.EDF.1 Interface Definition

3.UI.EDF.1.1 Exported Facilities

Initialization Functions

Name	Parameters	Constraints
+source+	p1:[source];O_ret	
	creates a source object p1 for editing and formatted output.	
+open_source+	p1:[source];I p2:[displ_typ];I p3:[source_id];O_ret	
	provides an identifier p3 for unique edit/format access to data source p1 where output will be in the form (character or image) indicated by p2; positions the !displ origin! for p3 at the first !point focus! in p1.	
+close_source+	p1:[source_id];I p2:[source];O_ret	
	terminates an active edit/format access to source identified by p1 and returns the source in its current state.	

Edit Functions

Name	Parameters	Constraints
+g/s_extent+	p1:[source_id];I p2:[TYP integer];O_ret/I	
	returns/sets the value of !displ extent!.	
+shift_extent+	p1:[source_id];I p2:[TYP integer];I	
	repositions the !displ origin! of source p1 to a !point focus! positioned at (approximately) p2 !displ extent!s from its current position.	
+mv_focus+	p1:[source_id];I p2:[CRT offset];I	
	makes !focus! of source p1 into a !point focus! and moves it to offset p2 from the current !displ origin! of p1.	

```
+expand_focus+          p1:[source_id];I
                        p2:[CRT offset];I
```
expands !focus! of source p1 so that it has an endpoint at offset p2 from the !displ origin! of p1.

```
+g_offset_posn+         p1:[source_id];I
                        p2:[TYP integer];I_opt
                        p3:[char_pos];I
                        p4:[position];O_ret
```
determines the position p4 within p1 of a !point focus! before character position p3 of a line which is p2 lines from the current start position of !focus!.

```
+g/s_edit_hold+         p1:[source_id];I
                        p2:[TYP displ_obj];O/I
```
returns/replaces the display object currently stored in the !edit hold! area for source p1.

```
+insert+                p1:[source_id];I
```
modifies the text contained in source p1 such that the contents of !edit hold! is inserted starting at the current !focus! location in p1; if !focus! is not a !point focus!, the contents of !edit hold! and !focus! are swapped.

```
+delete+                p1:[source_id];I
```
modifies source p1 such that the character string identified by !focus! is deleted, replacing the value of !edit hold! for p1.

```
+copy+                  p1:[source_id];I
```
makes a copy of the text in source p1 contained in !focus! and stores this text as the new value of !edit hold! for p1.

```
+undo+                  p1:[source_id];I
```
reverses the effect of the preceding insert, delete, or copy function applied to source p1.

```
+locate+                p1:[source_id];I
                        p2:[pattern];I
                        p3:[TYP boolean];O_ret
```
sets the position of !focus! to the position of the next occurrence (following the current position of !focus!) of text pattern p2 in source p1; p3 indicates whether the pattern was found.

```
+g_text+                p1:[source_id];I
                        p2:[TYP displ_obj];O_ret
```
returns !displ extent! (character or image) lines of the formatted display object form of the source text, such that the first line includes the start of !focus! in source p1.

Name	Parameters	Constraints
+g/s_page_length+	p1:[source_id];I p2:[TYP integer];0_ret/I	

defines the number of lines of text to be grouped as a page for source p1; a value of zero for p2 indicates that text will be continuous rather than paged.

+s_medium+	p1:[source_id];I p2:[TYP displ_medium];I	

causes !focus! of source p1 to have all defined attributes of display medium p2 (undefined attributes of p2 do not affect the attributes of the !focus! of p1).

+reset_medium+	p1:[source_id];I	

resets the display medium attributes of the !focus! of source p1 to be the same as its enclosing context.

+g/s_margins+	p1:[source_id];I p2:[line_area];0_ret/I	%fragmenting lines%

returns/defines the line area p2 of each line of text in the !focus! of source p1.

+g/s_align+	p1[source_id];I p2[alignment];0_ret/I	%fragmenting lines%

returns/defines the alignment of text lines in the !focus! of source p1.

+g/s_justify+	p1:[source_id];I p2:[TYP boolean];0_ret/I	%fragmenting lines%

returns/defines whether text lines in the !focus! of source p1 should be right justified using variable spacing (p2 = $TRUE$) or not (p2 = $FALSE$).

3.UI.EDF.1.2 Local Dictionary

[alignment]	[TYP enum: $left$, $center$, $right$].
[char_pos]	a [TYP integer] identifying a [position] relative to the start (if positive) or to the end (if negative) of a line of text such that 0 is before the first character of a line and -1 is after the last character.
!displ extent!	the number of lines obtainable as a unit with one access for display of a text source.
[displ_typ]	[enum : $char$, $image$].

!edit_hold!	an internal repository for temporary storage of edit data for a source.
!focus!	two [position]s (endpoints) within a data source that define a current focus of interest as the data between the [position]s.
[format_id]	a unique identifier for a set of format characteristics.

%fragmenting lines%

[line_area]	a [TYP rec] of (1) a [TYP integer] indicating the left alignment of the area relative to the [line_area] of any preceding text lines and (2) a [TYP integer] indicating the right alignment similarly.
[pattern]	?? [TYP charstr.pattern]
!point focus!	a !focus! whose origin and end point are at the same [position] in a data source.
[position]	a point between two adjacent character locations within a text data source; source start and end are two such points.
[source_id]	a unique identifier characterizing an active edit/format activity for a data source.
[unit_id]	an identifier for a character string delimited by two [position]s.

3.UI.EDF.1.3 Information Hidden

1. The internal representation of textual data; transformations required to modify this data and to display it under formatting guidelines.

3.UI.EDF.2 Design Support

3.UI.EDF.2.1 Interface Assumptions

1. All user visible data must be presented either in a symbolic or a textual form. Symbolic forms are defined monolithically to correspond to a single value of some user concept. A different value is displayed by replacing the symbol by another symbol. A textual representation of a concept's value differs in that value changes may be indicated by a (partial) modification of the representation.

2. All textual data representation has two aspects: content and format. Edit functions are required for modification of content. Formatting functions allow definition of a mapping from the content to an external representation for display.

3. Editing as defined here has two effects: modification of the value of a textual data object and (potentially) modification of information displayed to a user. These differ due to the transformations determined by formatting. It must be possible to obtain a displayable excerpt of a text object under some format on demand.

3.UI.EDF.2.2 Design Issues

1. How can internal data presentation templates (see the External Forms module) be integrated into a text editing/formatting framework? Clearly it would be useful to be able to include formatted data into text documents and to allow integration of editing of that data and free text. It is not clear, however, the best way to do this. It may be necessary to merge this and the External Forms module.

3.UI.EDF.2.3 Implementation/Configuration Information

1. Descriptions of integrated, interactive editing/formatting systems in the references provides a model of the kind of facilities this module should provide. The discussion of issues concerning document formatting in section 3 of Reference 2 is particularly useful.

3.UI.EDF.2.4 References

1. N. Meyrowitz, A. van Dam. "Interactive Editing Systems: Part I", ACM Computing Surveys 14(3), September 1982, 321-352.

2. R. Furuta, J. Scofield, A. Shaw. "Document Formatting Systems: Survey, Concepts, and Issues", ACM Computing Surveys 14(3), September 1982, 417-472.

3. Proceedings of the ACM SIGPLAN SIGOA Symposium on Text Manipulation, SIGPLAN Notices (ACM) 16(6), June 1981.

3.UI.FRM External Forms (FRM) Module

The external forms module provides facilities for construction and use of external display representations of aggregate objects. These representations can be parameterized to allow filling with variable data before display.

3.UI.FRM.1 Interface Definition

3.UI.FRM.1.1 Exported Facilities

Template Definition Functions

Name	Parameters	Constraints
++template++	p1:[TYP type];I_opt p2:[templ_id];O_ret	
defines a display template p2 which has an !item id! of type p1 for data access.		
++format++	p1:[templ_id];I p2:[layout];I	
defines the layout p2 of subtemplates of p1.		
++subtemplate++	p1:[templ_id];I p2:[templ_id];O_ret	%%inval templ use%%
defines a subtemplate p2 of template p1.		
++label++	p1:[templ_id];I p2:[TYP displ_obj];I	%%inval templ use%%
defines template p1 to be a display object p2 used as a label.		
++item_id_constr++	p1:[templ_id];I p2:[TYP type];I p3:[func_id];I	
defines a function p3 for template p1 that provides an !item id! for template p1 subtemplates.		
++value++	p1:[templ_id];I p2:[ext_type];I	%%inval templ use%%
defines template p1 as displaying values of type p2.		

```
++value_source++        p1:[templ_id];I                    %%inval templ use%%
                        p2:[func_id];I                     %%value conflict%%
                        p3:[func_id];I
```

identifies function p2 as the source of data item values to be displayed in template p1; p2 has one input parameter, the !item id! associated with p1 (I_opt) for data item identification; function p3 formats an input value (of the type associated with p1) which is returned as a [TYP displ_obj] value.

```
++value_dest++          p1:[templ_id];I                    %%inval templ use%%
                        p2:[func_id];I                     %%value conflict%%
                        p3:[func_id];I
```

identifies function p3 that modifies internal data values of the type associated with template p1; p3 has two input parameters, the !item id! associated with p1 (I_opt) for data item identification and the output of p2 (I); function p2 accepts a [TYP charstr] representation of the data value (input associated with p1) which is returned as a value of the type associated with p1.

```
++value_constr++        p1:[templ_id];I                    %%value conflict%%
                        p2:[TYP type];I
                        p3:[func_id];I
```

defines template p1 to be a representation of a composite data item of type p2 constructed from the "value"s of p1's subtemplates using function p3; p3 must expect one correctly typed parameter for each subtemplate of p1 that has a value in the order of subtemplate definition.

```
++value_decomp++        p1:[templ_id];I                    %%value conflict%%
                        p2:[ext_type];I
                        p3:[func_id];I
```

defines subtemplate p1 to be a representation of a value of type p2 extractable by function p3 from the value of the template of which it is a component.

```
++select++              p1:[templ_id];I
                        p2:[func_id];I
```

defines a function p2 which, given an !item id! for template p1, responds to user "selection" of p1 in a display form.

```
++action++              p1:[templ_id];I
                        p2:[action_name];I
                        p3:[INP keybd];I
                        p4:[INP key];I
                        p5:[func_id];I
```

defines a generic action named by p2 associated with template p1 which can be referenced by key p4 in the definition of keyboard p3 to invoke function p5 with an !item id! parameter if p1 has one.

Display Form Functions

Name	Parameters	Constraints
+open_displ+	p1:[templ_id];I p2:[displ_type];I p3:[!item_id!_type];I_opt p4:[displ_id];O_ret	

initiates use of a display form p4 represented by template p1 and associated with a data item identified by p3; p2 determines the form in which p4 is to be displayed.

+close_displ+ p1:[displ_id];I
terminates use of display form p1.

+print+ p1:[displ_id];I
causes the display form p1 to be printed on a hardcopy printer.

+update+ p1:[displ_id];I
p2:[TYP charstr];I
causes the value function associated with a template at the !focus! of display form p1 to be invoked with the character string p2 converted to a value of the appropriate type.

+select+ p1:[displ_id];I
causes the select function associated with a template at the !focus! of display form p1 to be invoked.

+inv_[action_name]+ p1:[displ_id];I
causes the function associated with the named action and a template at the !focus! of display form p1 to be invoked.

+g_image+ p1:[displ_id];I
p2:[TYP displ_obj];O
obtains a display object p2 which is an external representation of display form p1.

3.UI.FRM.1.2 Local Dictionary

[action_name]	a [TYP charstr] uniquely representing a type of action associated with a template.
[displ_id]	a unique identifier for a display form which is an instance of a defined template for which data values can be determined.
[displ_type]	[TYP enum : $char$, $image$].

[ext_type]	for a template with subtemplates, any [TYP type]; for all other templates, any [TYP type] which has functions +g_extrep+ and +s_extrep+ defined.
!focus!	a position within a display form which is the current focus of all user inputs.
[func_id]	a [CFG func_name].
%%inval templ use%%	an attempt to characterize a template in more than one way as composed of subtemplates, as containing a label, or as containing a data value.
!item id!	a data value that uniquely identifies a data item associated for display and update with a display frame.
[!item id!_type]	the [TYP type] of an !item id! for a particular [templ_id].
[label]	a [TYP charstr] which is used to label a field in a form
[layout]	[TYP enum : $horiz$, $vert$] (the method of aligning subtemplates within a template layout).
[templ_id]	a unique identifier for a display template.
[upd_func]	a [TYP func_id] and a [TYP seq] of [field_id]s that defines the actual parameters for the function; all fields used as parameters must be in subordinate templates of the template to which the function is attached.
%%update conflict%%	a template has been defined to have more than one associated internal value updating function.
%%value conflict%%	a template has been defined to have a value obtainable in more than one way.

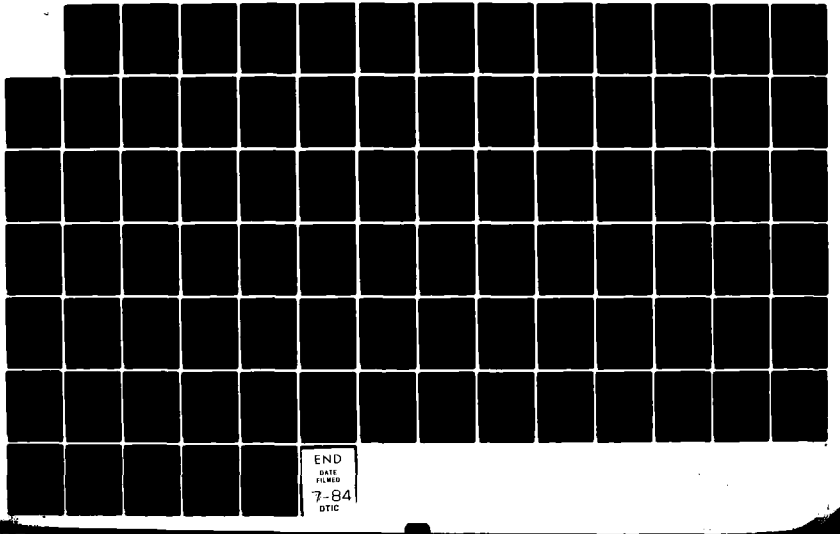
3.UI.FRM.1.3 Information Hidden

1. How display templates are represented and manipulated.

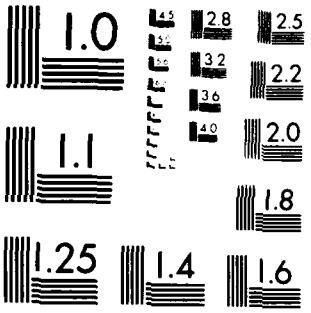
2. How display forms are constructed from template definitions and formatted internal data; when and how internal data is obtained and formatted for use in a form.

AD-A141 910 SOFTWARE ACQUISITION MANAGER'S WORKSTATION (SAM/WS) SYSTEM DESIGN(U) SOFTWARE ARCHITECTURE AND ENGINEERING INC ARLINGTON VA G H CAMPBELL ET AL. 30 APR 84 2/2

UNCLASSIFIED SAE-DC-84-R-004 N00014-82-C-0428 F/G 9/2 NL



END
DATE FILMED
7-84
DTIC



3. How input values are correlated to a particular subtemplate area of a display form and used to initiate an action or to modify internal data.

3.UI.FRM.2 Design Support

3.UI.FRM.2.1 Interface Assumptions

1. External display of information can be viewed as the display of either a composite template constructed from subtemplates or a data template containing a value formatted for display.

2. The subtemplates of a composite template can be layed out vertically (in a column) or horizontally (in a row). Further composition of composite templates supports general layout requirements.

3. A template which has no subtemplates can have either a typed data value or a fixed label associated for display. A template which has subtemplates that can be given values can be defined to have a value constructed by some defined function from the sequence of its subtemplate values.

4. A template can be selected by a user from the display to indicate the invocation of some action. A function can be defined and associated with the template which provides the meaning of the action intended by the user.

5. Access to internal data values require the identification of functions that can be used to obtain and modify those values. Since internal data values can have arbitrary type, functions for the conversion between these values and external representations ([TYP displ_obj] on output and [TYP charstr] on input) must also be identified.

6. Since many data entities could be displayed using a single template definition, functions invoked for data access or other actions must receive an identifier for the particular entity being manipulated and apply its action properly.

7. The definition of a template (and its associated subtemplates) is sufficient to derive the external representation of a filled data form to be displayed on physical media as long as the form (character or image) of display objects expected by that media is known.

3.UI.FRM.2.2 Design Issues

1. How to allow dynamically varying number of subtemplates for a template? (e.g., for a user constructed diagram or data structure that has a variable number of components)

2. How to determine when to get new values to fill a display form? (e.g., whenever +update+ is called and periodically otherwise while a form is in use)

3. Order of function execution when both a template and a subtemplate have associated function attachments?

4. Whether/how to allow sharing of subtemplate definitions by independent templates? (and avoid self reference)

5. How to manage value construction when more than one user input is needed to construct a valid value? (subtemplates that together constitute a single internal value)

3.UI.FRM.2.3 Implementation/Configuration Information

1. Reference 1 describes a system with many facilities similar to those required for this module. That system is more limited in some ways and more general in others but provides a good model of what this design attempts.

3.UI.FRM.2.4 References

1. Richard E. Fikes, "Odyssey: A Knowledge-Based Assistant", Artificial Intelligence 16 (3), July 1981, 331-361.

2. Mary Shaw, Ellen Borison, Michael Horowitz, Tom Lane, David Nichols, Randy Pausch. "Descartes: A Programming-Language Approach to Interactive Display Interfaces" in Proceedings of the SIGPLAN '83 Symposium on Programming Language Issues in Software Systems (ACM SIGPLAN Notices 18(6)) June 27-29, 1983, 100-111.

3.AD.PKI Package Integration (PKI) Module

The package integration module allows the integration of separately developed programs into an application system. Such programs must exist in a form which is executable and have known interface requirements, both for external invocation of embedded functions and for embedded invocation of external functions.

3.AD.PKI.1 Interface Definition

3.AD.PKI.1.1 Exported Facilities

Name	Parameters	Constraints
++defn++	p1:[pkg_id];I p2:[CFG obj_prog];I p3:[TYP charstr];I	

defines a package p1 which represents an executable program p2 for which source code is not accessible; p3 describes the general capabilities of the package to aid in evaluating the applicability of the package.

++export++	p1:[pkg_id];I p2:[func_id];I p3:[TYP seq] of [func_parm];I p4:[TYP charstr];I	

defines a function p2 invocable in package p1 by other programs using parameter types identified by p3; p4 describes the purpose of the function sufficiently for correct use.

++import++	p1:[pkg_id];I p2:[ext_func_id];I p3:[func_id];I p4:[TYP seq] of [func_parm];I p5:[TYP charstr];I	

identifies a function p2 external to package p1 that the package invokes with the identifier p3 and parameters typed as specified by p4; p5 describes the expected function and assumptions about p2 sufficiently to justify its selection or future replacement.

3.AD.PKI.1.2 Local Dictionary

[ext_func_id]	a [LNG name] which distinguishes a [LNG function] that is invocable.

[func_id]	a [LNG name] attached to a function that is defined in a !package! and is accessible for execution by other programs.
[func_parm]	a [TYP lbl_setun] of a !TYP type! that characterizes the data type of a parameter of a function from the perspective of a !package! and a [mode] which determines how the parameter is accessed within the defining function.
[mode]	enumerated: IN, OUT, I/O, O_ret, IN_opt, OUT_opt
!package!	a set of programs which provide [LNG function]s for invocation by other programs and which may invoke [LNG function]s defined by other programs.
[pkg_id]	a [LNG name] which distinguishes a !package! from the set of all defined !package!s.

3.AD.PKI.1.3 Information Hidden

1. Mechanisms required for integration of separately developed program packages into an application system.

3.AD.PKI.2 Design Support

3.AD.PKI.2.1 Interface Assumptions

1. It is necessary to provide access to packages of programs that have been developed separately. It is sufficient that an executable form of a package be accessible if its external interfaces can be adequately described.

2. A package must define (i.e., export) at least one function that can be invoked externally to initiate the operation of programs in the package. A package may define any number of such functions. Every function defined within a package has a unique name that can be used to invoke it from outside the package. Each function has a fixed number of parameters whose types can be specified using the data typing terminology of the Abstract Data Type module.

3. Execution of a package's programs may depend on the availability of functions defined external to the package. Each such function has a unique name by which it is referenced within the package.

3.AD.PKI.2.2 Design Issues

1. Should exported or imported functions be allowed to have optional parameters or variable parameter types that are used to characterize overloaded functions? How can exported functions be distinguished after the translation from source into object code?

2. How can packages for which no source code is available be integrated into a SAM/WS? What information is needed (e.g., a symbol table that gives a program label-to-absolute address (relative to the start of the package object code) mapping)?

3.AD.PKI.2.3 Implementation/Configuration Information: None.

3.AD.PKI.2.4 References: None.

3.AD.EXP Expert System (EXP) Module

The expert system module provides facilities for the specification and use of application domain knowledge. Knowledge can be used to infer application object characteristics based on known characteristics. Supporting facilities allow the specification of domain metaknowledge that controls the use of domain knowledge and the justification of inferences made from this knowledge. An alternative use of domain knowledge is in the validation of existing object characteristics with the intent of identifying inconsistencies between known characteristics with respect to domain knowledge.

3.AD.EXP.1 Interface Definition

3.AD.EXP.1.1 Exported Facilities

Name	Parameters	Constraints
++KB++	p1:[OBJ domain];I p2:[KB];O_ret	
defines a !knowledge base! p2 in application domain p1.		
++relation++	p1:[KB];I p2:[inf_mech];I p3:[reln];I p4:[reln_id];O_ret	
adds a relation p4 described by p3 to !knowledge base! p1 that can be processed with inference mechanism p2.		
++g_reln_match++	p1:[KB];I p2:[reln_pattern];I p3:[TYP set] of ([reln_id]);O_ret	
identifies all relations p3 in !knowledge base! p1 whose definition matches pattern p2.		
++erase++	p1:[KB];I p2:[reln_id];I	
removes relation p2 from !knowledge base! p1.		
++descr++	p1:[KB];I p2:[reln_id];I p3:[TYP charstr];I	
provides a description p3 which explains the basis for relation p2 of !knowledge base! p1 in application domain terms.		

```
+infer+                 p1:[OBJ obj_id];I
                        p2:[TYP set] of ([OBJ attr_id]);I
```
initiates an attempt to infer values for attributes p2 of object p1 using relations defined for objects in the domain of p1 (a caller is delayed until the attempt is completed).

```
+validate+              p1:[domain];I
                        p2:[OBJ obj_id];I
                        p3:[OBJ attr_id];I
                        p4:[TYP set] of ([TYP lbl_setun] of (obj:[OBJ obj_id],
                        attr:[OBJ attr_id]));0_ret
```
attempts to identify any inconsistency between the value of attribute p3 of object p2 and other attribute values of application domain p1, based on relations defined in domain p1 !knowledge base!s; p4 indicates the set of attributes whose values are inconsistent with that of p3.

```
+justify+               p1:[OBJ obj_id];I
                        p2:[OBJ attr_id];I
                        p3:[TYP seq] of ([TYP lbl_setun] of
                        (type:[KB_type], KB:[KB], reln:[reln_id]));0_ret
```
identifies the relations p3 that were used to determine the value of attribute p2 of object p1.

```
+g_reln+                p1:[KB];I
                        p2:[reln_id];I
                        p3:[reln];0_ret
```
returns the relation named by p2 in !knowledge base! p1.

```
+g_reln_descr+          p1:[KB];I
                        p2:[reln_id];I
                        p3:[TYP charstr];0_ret
```
provides a description of the basis for relation p2 of !knowledge base! p1 in application domain terms.

3.AD.EXP.1.2 Local Dictionary

[and_cond]	a [TYP seq] of ([cond]).
[antec_cond]	a [TYP seq] of ([and_cond]) which defines alternative antecedent conditions, any one of which being true activates the consequent condition of the containing data relation.
[apply_actions]	a [TYP seq] of [LNG func_id] (?) that define a sequence of actions to take when a relation is satisfied.
[cntl_antec_cond]	a [TYP seq] of ([cntl_and_cond]) which defines alternative antecedent conditions, any one of which being true activates the consequent condition of the containing control relation.

[cntl_and_cond]	a [TYP seq] of ([cntl_antec_pred]).
[cntl_antec_pred]	
[cntl_conseq_cond]	a [cntl_conseq_pred] which defines the consequent condition of a control relation.
[cntl_conseq_pred]	
[cntl_reln]	[TYP lbl_setun] of (antec:[cntl_antec_cond], conseq:[cntl_conseq_cond], action:[apply_actions], confid:[confidence], expl:[explanation]).
[cond]	([OBJ attr_id], [pred], [expr]).
[confidence]	
[conseq_cond]	a [TYP seq] of ([TYP lbl_setun] of (cond:[and_cond], prob:[symb_prob])) which defines the consequent conditions of a relation.
[data_reln]	[TYP lbl_setun] of (antec:[antec_cond], conseq:[conseq cond], action:[apply_actions], confid:[confidence], expl:[explanation]).
[explanation]	a [TYP charstr] which gives an extended explanation of the rationale for a relation.
[expr]	
!knowledge base!	a set of relations that define logical relationships between object attribute values (within the framework of defined inference mechanisms).
[KB]	a !knowledge base!.
[pred]	[TYP enum : EQ, NE, LT, GT, LE, GE].
[reln]	[data_reln] or [cntl_reln].
[reln_id]	an identifier which uniquely identifies a relation within a !knowledge base!.
[reln_pattern]	
[symb_prob]	

3.AD.EXP.1.3 Information Hidden

1. How application domain knowledge is represented as relations.

2. How knowledge is used to infer new data values from known values.

3.AD.EXP.2 Design Support

3.AD.EXP.2.1 Interface Assumptions

1. An application domain is a collection of knowledge that describes relationships between entities within that domain. A knowledge base is a collection of descriptions that characterize relationships that are likely to be of interest together (e.g., relationships that describe how to determine the value of all attributes of a particular class of entity). Description knowledge (*referred to* as data relations) deal with inferring new data values from known values. Control knowledge (referred to as control relations) deal with determining the order in which data relations are investigated to satisfy specified goals. A knowledge base defines an agenda of relations to apply to the satisfaction of a goal. Data relations define inferences on object-associated data values; control relations define modifications to the agenda within which they are defined.

2. Data relations define logical relationships between entities characterized as abstract objects (via the abstract object module). Abstract objects are organized into classes, each of whose members is characterized by a collection of attributes that either have a typed value or refer to other objects. Data relations define valid relationships between attribute values. The abstract graph that defines which attribute values can be inferred from others is referred to as the attribute hierarchy. An attribute hierarchy constrains the legal inference relationships (i.e., potential data dependencies) between attributes such that a knowledge base is an instantiation of an abstract attribute hierarchy and a database is an instantiation of the knowledge bases comprising an application doma .

3. Inference relations have an external representati nd an explanation of meaning and context of use that is useful for j fying how and why particular data has been derived. These are necessary components of the definition of a relation and are appropriate both for data relations and control relations.

4. Just as inference relations can be used to derive unknown data values, the consistency of known data values can be determined by analysis of the validity of all relations that specify how those values are logically related. It is sufficient to support the validation of a single value against existing data values since a values in a collection cannot become invalid except by the addition of new values.

3.AD.EXP.2.2 Design Issues

1. How should knowledge bases be organized to best support inference focusing while maintaining independence of the organization of knowledge from its use? An application system may encompass knowledge of more than one application domain. A knowledge base for one domain should be independent of all other domains and of the organization of that knowledge. Within a domain, it should be possible to modify knowledge without modifying the way inferencing is invoked. This requires that knowledge bases be distinguished by domain but relationships between knowledge bases within a domain are hidden.

2. In some cases, it may be desirable to investigate the implications of assigning a particular value to an attribute before actually making the assignment. One alternative considered was to provide a facility for performing a "pre-justify" to determine what other attributes might be affected if a value were assigned. A better approach is to assume the possibility of making a "conjecture" of a value that could subsequently be either "confirmed" or "denied". This requires the

ability (in the abstract object module?) to establish a temporary context for object definition and value assignment in support of experimentation that can be easily discarded or made permanent, as appropriate.

3.AD.EXP.2.3 Implementation/Configuration Information

1. A side effect of inferring a data value should be the establishment of a "data dependency" between the inferred value and the values from which it was inferred. This allows rederivation of the inferred value if, at some future time, one of the supporting values changes. It also provides a trace, along with the relation support, for justifying how and why a particular value was derived. The abstract object module provides the facilities for recording data dependencies as well as actual values.

3.AD.EXP.2.4 References

1. Knowledge Engineering System (KES), General Description Manual, Software Architecture and Engineering, Inc., Arlington, VA 22209.

2. M. Stefik, et. al. "The Organization of Expert Systems: A Tutorial", Artificial Intelligence 18,2 (March 1982), 135-173.

3.AD.OBJ Abstract Object (OBJ) Module

The abstract object module provides for the definition of classes of objects, each element of which is characterized by the values of a set of characteristic attributes. An attribute, in turn, may be a reference to another object or it may be a typed data value. Relating one object to others via an attribute allows contextual rather than named references to those objects.

3.AD.OBJ.1 Interface Definition

3.AD.OBJ.1.1 Exported Facilities

Name	Parameters	Constraints
++class++	p1:[domain];I p2:[LNG name];I	
defines a class of objects named by p2 in application domain p1.		
++subset++	p1:[obj_typ];I p2:[LNG name];I	
defines a class of objects named by p2 which is a subset of the objects in class p1.		
++attr_value++	p1:[obj_typ];I p2:[LNG name];I p3:[TYP type];I	
defines for objects in class p1 an attribute p2 of type p3.		
++attr_obj++	p1:[obj_typ];I p2:[LNG name];I p3:[obj_typ];I	
defines for objects in class p1 an attribute p2 of type [object] in object class p3.		
++key++	p1:[obj_typ];I p2:[TYP set] of [attr_id];I	
indicates that the values of attributes p2 uniquely characterize each object in class p1 (i.e., any value which is a composite of the values of attributes in p2 uniquely identifies either zero or one (potential) member in object class p1).		

```
++descr++               p1:[obj_typ];I
                        p2:[attr_id];I
                        p3:[VC charstr];I
```
provides a description of attribute p2 of object class p1 which explains the meaning and use of that attribute in the context of the application system definition.

```
++view++                p1:[obj_typ];I
                        p2:[view_id];I
                        p3:[FRM templ_id];I
```
defines a view attribute p2 (with a [TYP displ_obj] value) of object class p1 to be derived from display template p3.

```
++value_rqst++          p1:[obj_typ];I
                        p2:[attr_id];I
                        p3:[FRM templ_id];I
```
identifies a display template p3 appropriate for requesting the value of attribute p2 of objects in class p1 from a user.

```
+classify+              p1:[obj_typ];I
                        p2:[object];I_opt/O_ret
```
defines an !object! p2 as a member of object class p1 and of all classes of which p1 is a subclass; if p2 is not input, an !object! is created and returned for later use.

```
+in_class+              p1:[object];I
                        p2:[obj_typ];I
                        p3:[VC boolean];O_ret
```
p3 = $TRUE$ indicates that p1 is a member of class p2.

```
+g_domain+              p1:[object];I
                        p2:[TYP set] of [domain];O_ret
```
identifies the domains p2 of which object p1 is a member.

```
+g_class+               p1:[object];I
                        p2:[TYP set] of [obj_typ];O_ret
```
identifies the classes p2 of which object p1 is a member.

```
+forget+                p1:[object];I                       %obj referenced%
                        p2:[obj_typ];I_opt
```
causes object p1 to be forgotten; if p2 is input, only attributes characteristic of class p2 are forgotten, making p1 no longer a member of that class.

```
+derive+                p1:[object];I
                        p2:[WIN win_id];I
                        p3:!TYP list! of [attr_id];I_opt
```
causes the values of attributes of object p1 to be derived through a combination of logical inferences and user input prompting via window p2 (in that order); values are derived only for attributes which have unknown value; if p3 is input, this is further restricted to those attributes except for others needed in support of ones included in p3.

```
+rqst_[attr_id]+        p1:[object];I
                        p2:[WIN win_id];I
```
causes the value of the indicated attribute of object p1 to be requested from the user in window p2 (a caller is delayed until a response is received).

```
+display+               p1:[object];I
                        p2:[view_id];I
                        p3:[WIN win_id];I
```
causes view p2 of object p1 to be displayed in window p3 for appropriate user action.

```
+add/rem_[attr_id]+     p1:[object];I
                        p2:[attr_val];0_ret/I
```
adds/removes a value p2 of an attribute of !object! p1.

```
+g/s_[attr_id]+         p1:[object];I
                        p2:[TYP set] of [attr_val];0_ret/I
```
returns/sets the value(s) p2 of an attribute of !object! p1.

```
+await_[attr_id]+       p1:[object];I
```
delays the caller until the value of the named attribute of object p1 is next set.

```
+select+                p1:[obj_typ];I
                        p2:[TYP set] of ([TYP lbl_setun] of
                         ("[attr_id]":[attr_val]));I_opt
                        p3:[TYP set] of [object];0_ret
```
identifies a set of objects p3 in class p1 with attribute values given by p2; if p2 is not input, all objects in class p1 are identified.

```
+intersect+             p1:[TYP set] of ([obj_typ]);I
                        p2:[TYP set] of ([object]);0
```
identifies a set of objects p2 which are members of all of the object classes identified in p1.

3.AD.OBJ.1.2 Local Dictionary

[attr_id]	a [LNG name] which distinguishes an !attribute! of objects in a given object class.
[attr_val]	[TYP lbl_setun] of (val:#, srce:[value_source], confid:[confidence]), where # is the attribute's type.
!attribute!	a discrete characteristic of an !object!.
[confidence]	the confidence the source of a data value has in the correctness of the value; a [TYP real] in the range from -1.0 to 1.0, where -1.0 indicates impossibility, 1.0 indicates certainty, and 0.0 indicates a randomly selected value.
[domain]	a [LNG name] which characterizes an application domain of object classes.
[obj_typ]	a [LNG name] which distinguishes a class of objects within a [domain] which have the same attribute structure.
[object]	a representation of an !object!.
!object!	a distinguishable entity in some application domain.
[user]	an [object] which represents an application system user (an object class).
[value_source]	a [TYP union] of ([user], [LNG prog_name], [EXP reln id], to indicate the source of a data value.
[view_id]	a [LNG name] for a description of an external representation of a user view of an object in a given object class.

3.AD.OBJ.1.3 Information Hidden

1. The representation of objects and attributes.

3.AD.OBJ.2 Design Support

3.AD.OBJ.2.1 Interface Assumptions

1. An abstract object orientation provides a framework for defining fixed, structural knowledge of an application domain and for describing object instances that have known (but changable) characteristics.

2. An object of an application domain can be characterized by attributes that "completely" define all knowable information about that object. Similar objects have the same attributes so that they can be viewed abstractly as a "class" of objects. Some objects in a class may be described in more detail by the specification of additional attributes. Similar objects within a class have the same additional attributes so that they can be viewed abstractly as a "subset" of the containing class of objects.

3. A useful abstract concept is that of "relationships" between objects. A relationship is equivalent to an attribute with the added characteristic that the value of the attribute is an object in some class of objects.

4. In addition to a value determined by the application domain, all attributes have other information associated. This includes a description that explains the meaning and use of the attribute and a form in which values can be requested from users. In addition, object classes have associated data display templates that define how attributes should be displayed together to users.

3.AD.OBJ.2.2 Design Issues

1. How should relationships be represented? How should attributes of relationships (as opposed to attributes of role participants) be supported? Explicit facilities for defining relationships could be provided but facilities are not necessary for both attributes and relationships: either can be defined in terms of the other. Given a foundation and perspective of abstract data typing for basic data values, the attribute approach seems more natural. Using attributes, a relationship can be represented in either of two ways: in the simple case, one object is viewed as an attribute of another such that a relationship exists from the first to the second (an inverse relationship can be defined from the second to the first but no explicit connection is made between these relationships); in the general case, an object class can be defined whose members have one attribute for each "role" in relationships of that type (the value of which is some object in an appropriate class) and other attributes that record information about the relationship (as opposed to about a particular role object).

2. functionally defined attribute values are the responsibility of the expert system module (inferences from known values are required). This is also true for inheritance of (default) values as opposed to inheritance of attribute slot definitions.

3. The existence of a "default" value for an attribute in some object class involves application domain knowledge. In the simplest case, a default is a relation concerning a single attribute that asserts that, if no other value is known, a particular value may be assumed. Traditionally, only this case is supported. By considering defaults to be an expert system responsibility, more complex cases can be

supported, such as having the default value vary depending on other attribute values. In addition, this makes it the responsibility of the expert system as to when a default value should be assumed insteading of assuming the value is unknown (undefined?) until a user provides a value or one can be derived.

4. How to provide for temporary contexts for objects? What about changes to objects in a context from outside the context (other users)?

5. provide for abstract operations/predicates on objects?

6. "copy" versus "reference" viewpoint on access to [object]s. (does access return a copy of an object or a pointer to internal storage? how to make shared access seem reasonable without revealing this)

7. How can the object view definition take advantage of views defined for a containing object class? Should a facility be provided to allow a view of an object class subset to be defined as an extension of a view of the object class? While this is a useful capability, it seems simpler to have it implemented by a "higher level" module. Identification of a simple way to have this module do it could change this decision.

8. What semantic concepts should an "object" module support? Three general concepts are provided: classification (via the object class concept), specialization (the inverse of generalization) (via the subset concept), and aggregation (via the attribute concept).

3.AD.OBJ.2.3 Implementation/Configuration Information: None.

3.AD.OBJ.2.4 References

1. D. C. P. and J. M. Smith. "Conceptual Database Design".

3.GE AS SAM General Expert Module (GE)

3.GE.PDA Project Domain Entry/Exit Module (PDA)

The PDA module activates a user session during which operations can be carried out on a project domain through the actions of other application software modules. During the activation (or signon) action the PDA module verifies that the specified project domain exists and that the user is authorized to access it. If the user has so specified, a new project domain is created. When requested, the PDA module deactivates the session, precluding further operations on the project domain until a subsequent session activation.

Associated with each project domain is a project user list that identifies those users who are authorized to operate within the project domain. A restricted subset of those users are empowered to modify the project user list through the facilities of the PDA module.

3.GE.PDA.1 Function Definition

3.GE.PDA.2 Design Support

3.GE.CDF Context Definition Module (CDF)

The CDF module sets the context of the user session by providing the facilities for defining and referencing versions of products in the project domain for which the session has been initiated. Following initiation of a session or whenever a change of the context in which products are being developed is required, the CDF module will define a new version set or select one from existing sets associated with the current project domain.

When requested, the CDF module will display the status of products in the project domain or, if a context has been established, in a version.

3.GE.CDF.1 Function Definition

3.GE.CDF.2 Design Support

3.GE.PDV Product Development Module (PDV)

The PDV module acts as a controller of the specialist modules of the Acquisition Requirements Definition and Acquisition Package Development modules. It does this by enabling a specialist module when requested. When enabled, the specialist module's prior context is restored and it is allowed to accept action requests. The PDV module allows no more than one specialist to be active at any time. Thus, before the services of another specialist can be obtained, the currently active specialist must be suspended. The PDV module accomplishes this by blocking the action requests to the specialist module being disabled and saving its context for a possible later reactivation.

The PDV module may also be requested to cancel an active specialist, in which case it directs the specialist to delete the product it is working on before disabling it.

The PDV module provides facilities for copying products from other version sets and for displaying products from the current or other version sets.

3.GE.PDV.1 Function Definition

3.GE.PDV.2 Design Support

3.GE.TUT Tutorial Assistance Module (TUT)

The TUT module displays tutorial information of two types: workstation and acquisition process. The type of information to be displayed is requested of the module and will be based on models of the workstation and the acquisition process. The module supports traversal through multiple tutorial display segments. Unless the request for a tutorial is specified as being in context, the module begins its traversal at the initial display segment and allows the requestor to follow various paths through the entire tutorial.

When the tutorial has been requested to be in context, the module employs the record of specialist activities to tailor the scope of tutorial information available to the requestor to that which is pertinent to current operations.

3.GE.TUT.1 Function Definition

3.GE.TUT.2 Design Support

3.GE.UTL Utility Services Module (UTL)

The UTL module provides facilities to archive the current project domain and to edit and print the current product.

3.GE.UTL.1 Function Definition

3.GE.UTL.2 Design Support

3.AR AS Acquisition Requirements Definition (AR) Module

3.AR.PSS Applicable Policies and Standards Specialist (PSS) Module

The Applicable Policies and Standards specialist (PSS) module supports the generation of an informal product consisting of a list of DoD, Navy, and NAVSEA policies and standards which apply to the acquisition package being developed. The specialist obtains information that characterizes the software product and its acquisition constraints. The information is used to draw inferences about policies and standards from rules that govern the software acquisition process. The inferences determine the list of applicable policies and standards.

When the list has been generated, the specialist module provides facilities to obtain relevant portions of the list, display or print the list, display justifications for the presence of particular elements on the list, display textual elaborations for particular elements of the list, as well as facilities to read and write the list on auxiliary storage and to delete the list.

3.AR.PSS.1 Function Definition

3.AR.PSS.1.1 Actions

The applicable policies and standards specialist module operates as a process that performs actions when presented with a stimulus in the form of new or modified data items. These actions may result in a change or refinement to the applicable policies and standards object and/or a change to the applicable policies and standards status.

Action	Condition	Data Item	Response
+cr_aps+	%null%	[obj_id]	%incomplete%

Establishes an applicable policies and standards object. The applicable policies and standards object is identified by [obj_id].

+gen_aps+	%incomplete%	[aps_attr] [obj_id]	%generated%

Refines the applicable policies and standards object identified by [obj_id]

by generating the applicable policies and standards list. The specialist module generates the initial applicable policies and standards list by obtaining the attributes of the software product and its acquisition constraints and then using these and other general attributes and rules to infer the contents of the list.

+mod_aps+ %generated% [edit_object] (%incomplete% OR [obj_id] %generated%)

Refines the generated applicable policies and standards object identified by [obj_id] by acquiring one or more data items to set or change corresponding elements of the applicable policies and standards object. If a data item changes the value of an attribute upon which the value of another entry in the applicable policies and standards object depends, the specialist module responds with %incomplete% to force regeneration of those portions of the list that depend on the attribute whose value has changed. When no data items are available, the applicable policies and standards specialist module waits for one or more to be made available. Entire list entries can be added or deleted by this action.

+get_list+ [receive_list] %generated% [obj_id]

Places an externally formatted instance of the list identified by [obj_id] into a dynamically obtained storage area represented by [receive_list]. If the state of the list is %null%, it is first generated. If the state of the list is %incomplete%, the generation of the list is completed before this action continues.

+get_specs+ [receive_specs] %generated% [obj_id]

Places an externally formatted instance of that portion of the list identified by [obj_id] that contains references to military specifications into a dynamically obtained storage area represented by [receive_specs]. If the state of the list is %null%, it is first generated. If the state of the list is %incomplete%, the generation of the list is completed before this action continues.

+get_stds+ [receive_stds] %generated% [obj_id]

Places an externally formatted instance of that portion of the list identified by [obj_id] that contains references to military standards into a dynamically obtained storage area represented by [receive_stds]. If the state of the list is %null%, it is first generated. If the state of the list is %incomplete%, the generation of the list is completed before this action continues.

+cancel_aps+ NOT %null% [obj_id] %null%

The applicable policies and standards object identified by [obj_id] is deleted.

+print_aps+ NOT %null% [obj_id]

An image of the applicable policies and standards object identified by [obj id] is printed.

+display_aps+ NOT %null% [obj_id]

An image of the applicable policies and standards object identified by [obj id] is displayed.

+display_jstfy+ NOT %null% [element_id]
[obj_id]

The justification for the choice of the pertinent element on the aps list identified by [obj_id] is displayed.

+display_elab+ NOT %null% [element_id]
[obj_id]

The textual elaboration, if available, of the pertinent element on the aps list identified by [obj_id] is displayed.

+write_aps+ NOT %null% [obj_id]

A copy of the applicable policies and standards object identified by [obj id] is transferred to the location in auxiliary storage addressed by the identification of the object. If a prior copy of the object had been made, it is deleted when the current copy is successfully completed.

+read_aps+ [read_id] %incomplete% or
[obj_id] %generated%

The copy of the applicable policies and standards object at a specified location in auxiliary storage is read by the applicable policies and standards specialist module. The location from which the object is read may be specified as either the current context or another context. In the former case, the effect is to read the most recently saved version of the applicable policies and standards object; in the latter case, the effect is to read a saved copy of an applicable policies and standards object from another acquisition package. The object that is read becomes the applicable policies and standards object of the current context identified by [obj_id] replacing the applicable policies and standards object which may have existed prior to the invocation of this action.

3.AR.PSS.1.2 Local Dictionary

Data item	Definition
[aps_attr]	the attributes describing software product characteristics and acquisition constraints needed by the applicable policies and standards specialist module to generate the applicable policies and standards list
[edit_object]	a data item that conveys an editing action to be performed on a product building block of the applicable policies and standards object
[element_id]	a data item that uniquely identifies an element of a generated applicable policies and standards list
[obj_id]	the identification of the object that represents the product being produced through the facilities of this specialist module; the identification is composed of [prod type] and [package_id]
[package_id]	the project identification and version identification of the acquisition package
[prod_type]	the type of product being produced by this specialist module; in this case the value of [prod_type] is "applicable policies and standards"
[read_id]	the identification of the applicable policies and standards object to be read from auxiliary storage
[receive_list]	the address of a storage area into which has been placed an externally formatted instance of the list
[receive_specs]	the address of a storage area into which has been placed an externally formatted instance of the portion of the list

containing the entries that reference military specifications

[receive_stds] the address of a storage area into which has been placed an externally formatted instance of the portion of the list containing the entries that reference military standards

%generated% the status of the applicable policies and standards object has been set to "generated", i.e., the attributes necessary for generating the list of the applicable policies and standards have been acquired and the applicable policies and standards list has been generated

%incomplete% the status of the applicable policies and standards object has been set to "incomplete", i.e., the applicable policies and standards object has been instantiated, but the acquisition of those attributes necessary for generating the list of the applicable policies and standards has not been completed

%null% an instance of an applicable policies and standards object for the current context does not exist

3.AR.PSS.1.3 Information Hidden

1. How the applicable policies and standards object is represented and stored

2. The implementation of actions on the applicable policies and standards object by the applicable policies and standards specialist module

3. The structure and content of the attributes and rules used by the specialist module to derive the list

4. The inference mechanism used to derive the list

3.AR.PSS.2 Design Support

3.AR.PSS.2.1 Interface Assumptions

3.AR.PSS.2.2 Design Issues

3.AR.PSS.2.3 Implementation/Configuration Information

3.AR.PSS.2.4 References

None.

3.AP AS Acquisition Package Development Modules

3.AP.DRS Contract Data Requirements List Specialist (DRS) Module

The CDRL specialist module supports the creation of a Contract Data Requirements List for an acquisition package. The specialist module uses a template to assemble a CDRL outline consisting of multiple formatted entries. The template supplies both the initial structure and the initial content of the CDRL outline. The content of each entry of the outline is provided from literal text strings and from information derived from product characteristics. In the latter case, the template guides the specialist module in acquiring the information on product characteristics. The specialist module acquires further information as it becomes available to add, delete, and modify the text used to form the CDRL. At any time following the initial generation of the CDRL, the specialist module will generate a schedule for submission of deliverables and insert the appropriate submission information into each entry. If, after a schedule has been generated, information bearing on the schedule is modified, the specialist module regenerates the schedule.

3.AP.DRS.1 Function Definition

3.AP.DRS.1.1 Actions

The CDRL specialist module operates as a process that performs actions when presented with a stimulus in the form of new or modified data items. These actions may result in a change or refinement to the CDRL object and/or a change to the CDRL status.

Action	Condition	Data Item	Response
+cr_cdrl+	%null%	[obj_id]	%incomplete% AND NOT %sched%

Establishes a CDRL object by creating a CDRL instance appropriate to the user's requirements. The new CDRL object is identified by [obj_id].

+gen_cdrl+	%incomplete%	[cdrl_char] [obj_id]	%generated% AND NOT %sched%

Refines the CDRL object identified by [obj_id] by generating the CDRL

outline. The specialist module generates the initial CDRL outline by assembling the product building blocks sequentially from the CDRL template. When it encounters a product building block that requires derivation of information from the product characteristics the specialist module acquires the needed data item and performs that function. The product characteristics obtained by the specialist module govern the number of CDRL entries in the generated outline.

+mod_cdrl+	%generated%	[edit_object] [obj_id]	(%incomplete% OR %generated%) AND (%sched% OR NOT %sched%)

Refines the generated CDRL object identified by [obj_id] by acquiring one or more data items to set or change corresponding elements of the CDRL object. If a data item changes the value of a product characteristic upon which the value of another entry in the CDRL depends, the specialist module responds with %incomplete% to force regeneration of those portions of the outline that depend on the product characteristic whose value has changed. When no data items are available, the CDRL specialist module waits for one or more to be made available. Entire CDRL entries can be added or deleted by this action.

+skd_cdrl+	%generated% AND NOT %sched%	[precedence] [rel_del] [obj_id]	%sched%

The specialist module obtains the precedence relationships of the deliverables, [precedence], and the end point of the activity relative to the date of contract award that produces each deliverable, [rel_del]. The specialist module validates the information and uses it to complete the CDRL object identified by [obj_id] by inserting submission dates into the entries. This action of the specialist module sets and maintains page headings, footings, and numbers for the cdrl object.

+cancel_cdrl+	NOT %null%	[obj_id]	%null%

The CDRL object identified by [obj_id] is deleted.

+print_cdrl+	NOT %null%	[obj_id]

An image of the CDRL object identified by [obj_id] is printed.

+display_cdrl+	NOT %null%	[obj_id]

An image of the CDRL object identified by [obj_id] is displayed.

+write_cdrl+	NOT %null%	[obj_id]

A copy of the CDRL object identified by [obj_id] is transferred to the location in auxiliary storage addressed by the identification of the object. If a prior copy of the object had been made, it is deleted when the current copy is successfully completed.

+read_cdrl+		[read_id] [obj_id]	%incomplete% or %generated%

The copy of the CDRL object at a specified location in auxiliary storage is read by the CDRL specialist module. The location from which the object is read may be specified as either the current context or another context. In the former case, the effect is to read the most recently saved version of the CDRL object; in the latter case, the effect is to read a saved copy of a CDRL object from another acquisition package. The object that is read

becomes the CDRL object identified by [obj_id] of the current context replacing the CDRL object which may have existed prior to the invocation of this action.

3.AP.DRS.1.2 CDRL Document Template

The template used by the CDRL specialist module to generate a CDRL outline is described in this section. The CDRL template guides the specialist module in generating a CDRL outline and in making modifications to the CDRL object in response to editing actions. A template is composed of uniquely identified product building blocks. Certain of the product building blocks contain literal text strings and will appear in the generated outline as they are shown in the template. Others contain data items bracketed with "@". These data items are derived from product characteristics acquired by the specialist module while generating the outline. The identifiers of blocks containing derived information are denoted with a suffix of "@".

The template is derived from the skeleton CDRL specified in appendix C of [SAM rqmt]. The outline generated by the specialist module will be identical to that skeleton CDRL with the addition of the actual values for the data derived from product characteristics. When the CDRL is generated from this template, the two blocks labelled cdrl_hdl@ and cdrl_hd2@ are used to produce a heading at the top of each page while the two blocks labelled cdrl_tr@ and cdrl_pg are used to produce a footing at the bottom of each page. Each page of the CDRL will contain, in addition to the heading and footing, one or more entries, each consisting of the blocks cdrl_fl@ through cdrl_fl6. Each block will be laid out in the entry in accordance with the format shown in appendix C of [SAM rqmt]. The page numbers in cdrl_pg will be maintained by +skd_cdrl+.

Block Id	Block
cdrl_hdl@	CONTRACT DATA REQUIREMENTS LIST
	ATCH NBR @nbr@ TO EXHIBIT @exh@
	CATEGORY @cat@
	TO CONTRACT/PR @contractno@
	1. SEQUENCE NUMBER / 2. TITLE OR DESCRIPTION OF DATA 3. SUBTITLE / 6. TECHNICAL OFFICE / 10. FRQNCY / 12. DATE OF 1ST SUBMISSI
	4. AUTHORITY (Data Item Number) / 5. CONTRACT REFERENCE / 7. DD250 REQ / 8. APP CODE / 9. INPUT TO IAC / 11. AS OF DATE / 13. DATE OF SBSQ SUBM/EVENT

Block Id	Block
cdrl_hd2@	SYSTEM/ITEM @program name@ CONTRACTOR @contractor@
	14. DISTRIBUTION AND ADDRESSEES (Addressees-Regular Copies/Repro Copies)
cdrl_f1@	1. @seqno@
cdrl_f2@	2. @title@
cdrl_f3@	3. @subtitle
cdrl_f4@	4. @authority
cdrl_f5@	5. @contractref@
cdrl_f6@	6. @techoffice@
cdrl_f7@	7. @DD250@
cdrl_f8@	8. @appcode@
cdrl_f9@	9. @iac@
cdrl_f10@	10. @frequency@
cdrl_f11@	11. @asofdate@
cdrl_f12@	12. @lstsub@
cdrl_f13@	13. @subsub@
cdrl_f14@	14. @dist@

Block Id	Block
cdrl_f15@	15. @total@
cdrl_f16	16. REMARKS
cdrl_tr@	PREPARED BY @prepby@ DATE @prepdt@ APPROVED BY @apprby@ DATE @apprdt@
cdrl_pg	PAGE ___ OF ___ PAGES

3.AP.DRS.1.3 Local Dictionary

Data item	Definition
[cdrl_char]	the product characteristics needed by the CDRL specialist module to generate the CDRL outline
[edit_object]	a data item that conveys an editing action to be performed on a product building block of the CDRL object
[obj_id]	the identification of the object that represents the product being produced through the facilities of this specialist module; the identification is composed of [prod type] and [package_id]
[package_id]	the project identification and version identification of the acquisition package
[precedence]	the required ordering of deliverables; i.e., the predecessor/successor relationships among the deliverables
[prod_type]	the type of product being produced by this specialist module; in this case the value of [prod_type] is "CDRL"
[read_id]	the identification of the CDRL object to be read from auxiliary storage
[rel_del]	the number of units of time following an event (e.g., contract award or delivery of a predecessor deliverable) that a data item will be delivered
@1stsub@	date of first submission of a deliverable; obtained or calculated by the specialist
@appcode@	CDRL field obtained by specialist

@apprby@	name of person approving CDRL; obtained by specialist
@apprdt@	date of approval of CDRL; obtained by the specialist
@asofdate@	CDRL field obtained or calculated by the specialist
@authority@	CDRL field obtained by specialist
@cat@	CDRL field obtained by specialist
@contractno@	contract number for this acquisition; obtained by specialist
@contractor@	name of contractor to whom the CDRL is addressed; obtained by specialist
@contractref@	CDRL field obtained by specialist
@DD250@	CDRL field obtained by specialist
@dist@	CDRL field obtained by specialist
@exh@	CDRL field obtained by specialist
@frequency@	frequency of distribution of the data item; obtained by specialist
@iac@	CDRL field obtained by specialist
@nbr@	CDRL field obtained by specialist
@prepby@	name of preparer of CDRL; obtained by specialist
@prepdt@	date of preparation of CDRL; obtained by specialist
@program name@	the name of the program for which the subject of this software acquisition is being procured; obtained by specialist

@seqno@	data item sequence number maintained by the specialist
@subsub@	date of subsequent submission of a deliverable; obtained or calculated by the specialist
@subtitle@	CDRL field obtained by specialist
@techoffic@	CDRL field obtained by specialist
@title@	CDRL field obtained by specialist
@total@	total number of copies of the data item to be distributed; calculated or obtained by specialist
%generated%	the status of the CDRL object has been set to "generated", i.e., the product characteristics necessary for generating the outline of the CDRL have been acquired and the CDRL outline has been generated
%incomplete%	the status of the CDRL object has been set to "incomplete", i.e., the CDRL object has been instantiated, but the acquisition of those product characteristics necessary for generating the outline of the CDRL has not been completed
%null%	an instance of a CDRL object for the current context does not exist
%sched%	set on when a schedule has been generated by the specialist module; set off when the CDRL object is established or when a field of any entry of the CDRL affecting the schedule has been edited

3.AP.DRS.1.4 Information Hidden

1. How the CDRL object is represented and stored.

2. The implementation of actions on the CDRL object by the CDRL specialist module.

3.AP.DRS.2 Design Support

3.AP.DRS.2.1 Interface Assumptions

3.AP.DRS.2.2 Design Issues

3.AP.DRS.2.3 Implementation/Configuration Information

3.AP.DRS.2.4 References

None.

3.AP.RPS Request for Proposal Specialist (RPS) Module

The request for proposal specialist module supports the creation of a request for proposal for an acquisition package. The specialist module uses a template to assemble a request for proposal outline. The template supplies both the initial structure and the initial content of the request for proposal outline. The content of the outline is provided from literal text strings and from information derived from product characteristics. In the latter case, the template guides the specialist module in acquiring the information on product characteristics. The specialist module acquires further information as it becomes available to add, delete, and modify the text used to form the request for proposal.

3.AP.RPS.1 Function Definition

3.AP.RPS.1.1 Actions

The request for proposal specialist module operates as a process that performs actions when presented with a stimulus in the form of new or modified data items. These actions may result in a change or refinement to the request for proposal object and/or a change to the request for proposal status.

Action	Condition	Data Item	Response
+cr_rfp+	%null%	[obj_id]	%incomplete%

Establishes a request for proposal object. The request for proposal object is identified by [obj_id].

+gen_rfp+	%incomplete%	[rfp_char] [obj_id]	%generated%

Refines the request for proposal object identified by [obj_id] by generating the request for proposal outline. The specialist module generates the initial request for proposal outline by assembling the product building blocks sequentially from the request for proposal template. When it encounters a product building block that requires derivation of information from the product characteristics the specialist module acquires the needed data item and performs that function.

+mod_rfp+	%generated%	[edit_object] [obj_id]	%incomplete% or %generated%

Refines the generated request for proposal object identified by [obj_id] by acquiring one or more data items to set or change corresponding elements of the request for proposal object. If a data item changes the value of a product characteristic, the specialist module responds with %incomplete% to

force regeneration of those portions of the outline that depend on the product characteristic whose value has changed. When no data items are available, the request for proposal specialist module waits for one or more to be made available.

+cancel_rfp+ NOT %null% [obj_id] %null%
The request for proposal object identified by [obj_id] is deleted.

+print_rfp+ NOT %null% [obj_id]
An image of the request for proposal object identified by [obj_id] is printed.

+display_rfp+ NOT %null% [obj_id]
An image of the request for proposal object identified by [obj_id] is displayed.

+write_rfp+ NOT %null% [obj_id]
A copy of the request for proposal object identified by [obj_id] is transferred to the location in auxiliary storage addressed by the identification of the object. If a prior copy of the object had been made, it is deleted when the current copy is successfully completed.

+read_rfp+ [read_id] %incomplete% or
[obj_id] %generated%
The copy of the request for proposal object at a specified location in auxiliary storage is read by the request for proposal specialist module. The location from which the object is read may be specified as either the current context or another context. In the former case, the effect is to read the most recently saved version of the request for proposal object; in the latter case, the effect is to read a saved copy of a request for proposal object from another acquisition package. The object that is read becomes the request for proposal object identified by [obj_id] of the current context replacing the request for proposal object which may have existed prior to the invocation of this action.

3.AP.RPS.1.2 Request For Proposal Document Template

The template used by the request for proposal specialist module to generate a request for proposal outline is described in this section. The request for proposal template guides the specialist module in generating a request for proposal outline and in making modifications to the request for proposal object in response to editing actions. A template is composed of uniquely identified product building blocks. Certain of the product building blocks contain literal text strings and will appear in the generated outline as they are shown in the template. Others contain data items bracketed with "@". These data items are derived from product characteristics acquired by the specialist module while generating the outline. The identifiers of blocks containing derived information are denoted with a suffix of "@".

The template is derived from the skeleton request for proposal specified in appendix A of [SAM rqmt]. The outline generated by the specialist module will be identical to that skeleton request for proposal with the addition of the actual values for the data derived from product characteristics.

Block Id	Block
rfp_cl@	STANDARD FORM 33 1. CONTRACT NO. @contractno@
rfp_pid@	2. SOLICITATION NO. @procurement id@ ADVERTISED (IFB) NEGOTIATED (RFP)
rfp_date@	3. CERTIFIED FOR NATIONAL DEFENSE UNDER BDSA REG 2 AND OR DMS REG 1 RATING 5. DATE ISSUED @rfp date@
rfp_pr@	6. REQUISITION PURCHASE REQUEST NO. @purch rqst@
rfp_isu@	7. ISSUED BY Code @dodaad@ @issuer@ BUYER/SYMBOL @buyer name@ PHONE: @buyer phone@
rfp_ofr@	8. ADDRESS OFFER TO @offer to@

Block Id	Block
rfp_inst@ 9.	Sealed offers in original and @#copies@ copies for furnisning the supplies or services in the schedule will be received at the place specified in block 8, or if handcarried, in the depository located in @deposit@ until @deadline time@ local time @deadline date@. If this is an advertised solicitation, offers will be publicly opened at that time. CAUTION-LATE OFFERS: See pars. 7 and 8 of Solicitation Instructions and Conditions. All offers subject to the following: 1. The Solicitation Instructions and Conditions, SF-33A, @sf33a edition@ edition, which is attached or incorporated herein by reference. 2. The General Provisions, SF 32, @sf32 edition@ edition, which is attached or incorporated herein by reference. 3. The Schedule included herein and/or attached hereto. 4. Such other provisions, representations, certifications, and specifications as are attached to or incorporated herein by reference. (Attachments are listed in the Table of Contents) FOR INFORMATION CALL @information@ (no collect calls)

rfp_tl

TABLE OF CONTENTS

THE FOLLOWING CHECKED SECTIONS ARE CONTAINED IN THE CONTRACT

(X)	SEC		PAGE
		PART I - GENERAL INSTRUCTIONS	
	A	Cover Sheet	
	B	Contract Form and Representations, Certifications, and Other Statements of Offeror	
	C	Instructions, Conditions, and Notices to Offerors	
	D	Evaluation Factors for Award	
		PART II - THE SCHEDULE	
	E	Supplies/Services and Prices	
	F	Description/Specifications	
	G	Packaging and Marking	
	H	Deliveries or Performance	
	I	Inspection and Acceptance	
	J	Special Provisions	
	K	Contract Administration Data	
		PART III - GENERAL PROVISIONS	
	L	General Provisions	
		PART IV - LIST OF DOCUMENTS AND ATTACHMENTS	
	M	List of Documents, Exhibits, and Other Attachments	

Block Id	Block
rfp_c2@	PART I - GENERAL INSTRUCTIONS @contractno@
rfp_t2	SECTION A
rfp_t3	SECTION B - CONTRACT FORM AND REPRESENTATIONS, CERTIFICATION, AND OTHER STATEMENTS OF OFFEROR
rfp_t4	SECTION C - INSTRUCTIONS, CONDITIONS, AND NOTICES TO OFFERORS
rfp_t5	SECTION D - EVALUATION FACTORS FOR AWARD
rfp_c3@	PART II - THE SCHEDULE @contractno@
rfp_t6	SECTION E - SUPPLIES/SERVICES AND PRICES Item / Supplies/Services / Qty / Unit / Unit Price / Total Amount
rfp_t7	SECTION F - DESCRIPTION/SPECIFICATIONS
rfp_t8	SECTION G - PACKAGING AND MARKING
rfp_t9	SECTION H - DELIVERABLES OR PERFORMANCE
rfp_t10	SECTION I - INSPECTION AND ACCEPTANCE
rfp_t11	SECTION J - SPECIAL PROVISIONS
rfp_t12	SECTION K - CONTRACT ADMINISTRATION DATA
rfp_c4@	PART III - GENERAL PROVISIONS @contractno@
rfp_t13	SECTION L - GENERAL PROVISIONS The clauses checked below, except those marked with an asterisk (*) are hereby incorporated by reference with the same force and effect as if set forth in full. Those clauses marked with an asterisk are attached hereto in full text. All clauses hereby incorporated by reference may be found in Section VII of the Defense Acquisition Regulations (DAR). Copies of the DAR may be purchased from the Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402. The clauses listed below and preceded by an "x" in the block to the left are applicable to this contract. Clauses preceded by "N/A" are not applicable. (X) Title / Date / Reference
rfp_cls@	@clauses@
rfp_c5@	@contractno@ PART IV - LIST OF DOCUMENTS, EXHIBITS, AND OTHER ATTACHMENTS

Block Id	Block
rfp_t14	SECTION M - LIST OF DOCUMENTS, EXHIBITS, AND OTHER ATTACHMENTS This solicitation package consists of the following checked material: ( ) 3 Copies DD Form 1707, Information to Offerors, 1 February 1976 ( ) 3 Copies Invitation for Bids/Request for Proposal including Standard Form 33, Solications Offer and Award, March 1977 and Standard Form 33A, Solicitation, Instructions and Conditions, July 1977 ( ) 3 Copies List of Clauses Incorporated by Reference, Fixed Price Supply Contracts - Pages ___ thru___ ( ) 3 Copies Additional General Provisions Fixed Price Supply Contracts - Pages ___ thru ___ ( ) 3 Copies List of Clauses Incorporated by Reference, Fixed Price Research and Development Contracts - Pages ___ thru ___ ( ) 3 Copies Additional General Provisions Fixed Price Research and Development Contracts - Pages ___ thru ___ ( ) 3 Copies of Clauses Incorporated by Reference, Fixed Price Services Contracts - Pages ___ thru ___ ( ) 3 Copies Additional General Provisions Fixed Price Services Contracts - Pages ___ thru ___ ( ) 3 Copies List of Clauses Incorporated by Reference, Cost Reimbursement Contracts - Pages ___ thru ___ ( ) 3 Copies Additional General Provisions Cost Reimbursement Contracts - Pages ___ thru ___ ( ) 3 Copies List of Clauses Incorporated by Reference, Cost Reimbursement Supply Contracts - Pages ___ thru ___ ( ) 3 Copies Additional General Provisions Cost Reimbursement Supply Contracts - Pages ___ thru ___ ( ) 3 Copies List of Clauses Incorporated by Reference, Cost Services Contracts - Pages ___ thru ___ ( ) 3 Copies Additional General Provisions Cost Services Contracts - Pages ___ thru ___ ( ) 3 Copies List of Clauses Incorporated by Reference, Time and Material and Labor Hour Contracts - Pages ___ thru ___ ( ) 3 Copies Additional General Provisions Time and Material and Labor Hour Contracts - Pages ___ thru ___ ( ) 3 Copies DD Form 1423 Contract Data Requirements List, consisting of the following checked Exhibits: ( ) Exhibit A, dated _____; ( ) Exhibit B, dated _____; ( ) Exhibit C, dated _____; ( ) Exhibit D, dated _____; ( ) Exhibit E, dated _____; ( ) Exhibit F, dated _____; ( ) Exhibit G, dated _____; ( ) Exhibit H, dated _____; ( ) Exhibit J, dated _____; ( ) Exhibit K, dated _____; ( ) Exhibit L, dated _____; ( ) Exhibit M, dated _____; ( ) Exhibit N, dated _____; ( ) Exhibit P, dated _____; ( ) Exhibit Q, dated _____; ( ) Exhibit R, dated _____; ( ) Exhibit S, dated _____; ( ) Exhibit T, dated _____; ( ) Exhibit U, dated _____; ( ) Exhibit V, dated _____; ( ) Exhibit W, dated _____; ( ) Exhibit X, dated _____; ( ) Exhibit Y, dated _____; ( ) Exhibit Z, dated _____; ( ) 3 Copies DD Form 1664, Data Item Description(s), dated 1 June 1968 ( ) 3 Copies DD Form 254, Contract Security Classification Specification, dated _____

Block Id	Block
rfp_t14 cont'd	( ) 3 Copies DD Form 633, Contract Pricing Proposal ( ) 3 Copies DD 1660, Management Systems Summary List, dated _____ ( ) 3 Copies DD Form 1564, Pre-Award Patent Rights Documentation ( ) 1 Copy Specification _____ _____ (UNCLASSIFIED), dated _____ ( ) 1 Copy Specification _____ _____ (UNCLASSIFIED), dated _____
rfp_sow@	( ) 1 Copy Statement of Work For @program name@ Dated @sow date@

3.AP.RPS.1.3 Local Dictionary

Data item	Definition
[edit_object]	a data item that conveys an editing action to be performed on a product building block of the request for proposal object
[obj_id]	the identification of the object that represents the product being produced through the facilities of this specialist module; the identification is composed of [prod type] and [package_id]
[package_id]	the project identification and version identification of the acquisition package

[prod_type]	the type of product being produced by this specialist module; in this case the value of [prod_type] is "request for proposal"
[read_id]	the identification of the request for proposal object to be read from auxiliary storage
[rfp_char]	the product characteristics needed by the request for proposal specialist module to generate the request for proposal outline
@#copies@	number of copies of proposal
@buyer name@	buyer/symbol
@buyer phone@	telephone number of buyer
@clauses@	the set of clauses that are applicable to this contract that will be contained in Section L of the RFP
@contractno@	contract number for this acquisition
@deadline date@	date by which proposal must be received
@deadline time@	local time of day by which proposal must be received
@deposit@	location of depositary to which proposal may be handcarried
@dodaad@	code
@information@	a telephone number that can be used by respondants to obtain information concerning the solicitation
@issuer@	issuer of rfp

@offer to@	address to which offer is to be sent
@procurement id@	solicitation number
@program name@	the name of the program for which the subject of this software acquisition is being procured
@purch rqst@	requisition purchase request number
@rfp date@	the publication date of the request for proposal
@sow date@	the publication date of the statement of work
@sf32 edition@	the edition identification of the SF-32 that is attached or incorporated with this RFP
@sf33a edition@	the edition identification of the SF-33A that is attached or incorporated with this RFP
%generated%	the status of the request for proposal object has been set to "generated", i.e., the product characteristics necessary for generating the outline of the request for proposal have been acquired and the request for proposal outline has been generated
%incomplete%	the status of the request for proposal object has been set to "incomplete", i.e., the request for proposal object has been instantiated, but the acquisition of those product characteristics necessary for generating the outline of the request for proposal has not been completed
%null%	an instance of a request for proposal object for the current context does not exist

3.AP.RPS.1.4 Information Hidden

1. How the request for proposal object is represented and stored.

2. The implementation of actions on the request for proposal object by the request for proposal specialist module.

3.AP.RPS.2 Design Support

3.AP.RPS.2.1 Interface Assumptions

3.AP.RPS.2.2 Design Issues

3.AP.RPS.2.3 Implementation/Configuration Information

3.AP.RPS.2.4 References

None

3.AP.SPS Specification Specialist (SPS) Module

The specification specialist module supports the creation of one of four types of system specification for an acquisition package: a Type A System Specification, a Program Performance Specification (PPS), a Functional Operation Design (FOD) Document, or a System Operational Design (SOD) Document. The specialist module uses a template to assemble a specification outline of the appropriate type. The template supplies both the initial structure and the initial content of the specification outline. The content of the outline is provided from literal text strings and from information derived from product characteristics. In the latter case, the template guides the specialist module in acquiring the information on product characteristics. The specialist module acquires further information as it becomes available to add, delete, and modify the text used to form the specification.

3.AP.SPS.1 Function Definition

3.AP.SPS.1.1 Actions

The specification specialist module operates as a process that performs actions when presented with a stimulus in the form of new or modified data items. These actions may result in a change or refinement to the specification object and/or a change to the specification status.

Action	Condition	Data Item	Response
+cr_spec+	%null%	[obj_id]	%incomplete%

Establishes a specification object by creating a specification of a type appropriate to the user's requirements. The specification object is identified by [obj_id].

+gen_spec+	%incomplete%	[spec_char] [spec_type] [obj_id]	%generated%

Refines the specification object identified by [obj_id] by generating the specification outline. The specialist module generates the initial specification outline by assembling the product building blocks sequentially from the appropriate specification template. The appropriate template is determined by [spec_type]. When it encounters a product building block that requires derivation of information from the product characteristics the specialist module acquires the needed data item and performs that function.

+mod_spec+ %generated% [edit_object] %incomplete% or
[obj_id] %generated%

Refines the generated specification object identified by [obj_id] by acquiring one or more data items to set or change corresponding elements of the specification object. If a data item changes the value of a product characteristic, the specialist module responds with %incomplete% to force regeneration of those portions of the outline that depend on the product characteristic wnose value has changed. When no data items are available, the specification specialist module waits for one or more to be made available.

+cancel_spec+ NOT %null% [obj_id] %null%

The specification object identified by [obj_id] is deleted.

+print_spec+ NOT %null% [obj_id]

An image of the specification object identified by [obj_id] is printed.

+display_spec+ NOT %null% [obj_id]

An image of the specification object identified by [obj_id] is displayed.

+write_spec+ NOT %null% [obj_id]

A copy of the specification object identified by [obj_id] is transferred to the location in auxiliary storage addressed by the identification of the object. If a prior copy of the object had been made, it is deleted when the current copy is successfully completed.

+read_spec+ [read_id] %incomplete% or
[obj_id] %generated%

The copy of the specification object at a specified location in auxiliary storage is read by the specification specialist module. The location from which the object is read may be specified as either the current context or another context. In the former case, the effect is to read the most recently saved version of the specification object; in the latter case, the effect is to read a saved copy of a specification object from another acquisition package. The object that is read becomes the specification object identified by [obj_id] of the current context replacing the specification object which may have existed prior to the invocation of this action.

3.AP.SPS.1.2 Specification Document Templates

Each of the templates used by the specification specialist module to generate a specification outline are described in this section. The specialist module chooses one template for an acquisition package based on the value of the product characteristic [spec_type].

The specification template guides the specialist module in generating a specification outline and in making modifications to the specification object in response to editing actions. A template is composed of uniquely identified product building blocks. Certain of the product building blocks contain literal text strings and will appear in the generated outline as they are shown in the template. Others contain data items bracketed with "@". These data items are derived from product characteristics acquired by the specialist module while generating the outline. The identifiers of blocks containing derived information are denoted with a suffix of "@".

3.AP.SPS.1.2.1 Type A Specification Template

The Type A Specification template is chosen by the specification specialist module when [spec_type]=typea. The template is derived from the skeleton Type A system specification specified in appendix E of [SAM rqmt]. The outline generated by the specialist module will be identical to that skeleton specification with the addition of the actual values for the data derived from product characteristics.

Block Id	Block
aspc_date@	@spec date@
aspc_t1	SYSTEM SPECIFICATION FOR
aspc_nm1@	@system name@
aspc_t2	Prepared by
aspc_prep@	@preparer@

Block Id	Block

TABLE OF CONTENTS

	Section	Page
	1. Scope.....	
	2. Applicable Documents.....	
	2.1 Military Specifications.....	
	2.2 Military Standards.....	
	2.3 Other Publications.....	
	3. Requirements.....	
	3.1 System Definition.....	
	3.2 Characteristics.....	
	3.3 Design and Construction.....	
aspc_t3	3.4 Documentation.....	
	3.5 Logistics.....	
	3.6 Personnel and Training.....	
	3.7 Functional Area Characteristics.....	
	3.8 Precedence of Requirements.....	
	4. Quality Assurance Provisions.....	
	4.1 General.....	
	4.2 Quality Conformance Inspections.....	
	5. Preparation for Delivery.....	
	6. Notes.....	

aspc_hd@ @spec heading@

aspc_t4

SYSTEM SPECIFICATION
FOR

@system name@

aspc_nm2@ 1. Scope

This specification establishes the performance, design, development, and test requirements for the @system name@.

2. Applicable Documents

aspc_t5 The following documents of the issue in effect on this date of solicitation form a part of this specification to the extent specified herein.

aspc_t6 2.1 Military Specifications

aspc_spcs@ @mil specs@

aspc_t7 2.2 Military Standards

aspc_stds@ @mil standards@

aspc_t8 2.3 Other Publications

aspc_t9 3. Requirements

aspc_t10 3.1 System Definition

aspc_t11 3.1.1 Item Diagrams

aspc_t12 3.1.2 Interface Definition

Block Id	Block
aspc_t13	3.1.3 Major Component List
aspc_t14	3.1.4 Government Furnished Property List
aspc_t15	3.1.5 Government Loaned Property List
aspc_t16	3.2 Characteristics
aspc_t17	3.2.1 Performance
aspc_t18	3.2.2 Reliability
aspc_t19	3.2.3 Maintainability
aspc_t20	3.2.4 Transportability
aspc_t21	3.3 Design and Construction
aspc_t22	3.3.1 Processes and Parts
aspc_t23	3.3.2 Product Marking
aspc_t24	3.3.3 Workmanship
aspc_t25	3.3.4 Interchangeability
aspc_t26	3.3.5 Safety
aspc_t27	3.3.6 Human Performance/Human Engineering
aspc_t28	3.4 Documentation
aspc_t29	3.5 Logistics
aspc_t30	3.5.1 Maintenance
aspc_t31	3.5.2 Facilities and Facility Equipment
aspc_t32	3.6 Personnel and Training
aspc_t33	3.6.1 Personnel
aspc_t34	3.6.2 Training
aspc_t35	3.7 Functional Area Characteristics
aspc_t36	3.8 Precedence of Requirements
aspc_t37	4. Quality Assurance Provisions
aspc_t38	4.1 General
aspc_t39	4.1.1 Responsibility for Tests

Block Id	Block
aspc_t40	4.1.2 Special Tests and Examinations
aspc_t41	4.2 Quality Conformance Insptections
aspc_t42	5. Preparation for Delivery
aspc_t43	6. Notes

3.AP.SPS.1.2.2 PPS Template

The PPS template is chosen by the specification specialist module when [spec_type]=pps. The template is derived from the skeleton Program Performance Specification specified in appendix F of [SAM rqmt]. The outline generated by the specialist module will be identical to that skeleton specification with the addition of the actual values for the data derived from product characteristics.

Block ID	Block
pspc_date@	@spec date@
pspc_t1	PROGRAM PERFORMANCE SPECIFICATION FOR
pspc_nm1@	@program name@
pspc_t2	Prepared by
pspc_prep@	@preparer@

TABLE OF CONTENTS

	Section	Page
	1. Scope.....	
	1.1 Purpose.....	
	1.2 Mission.....	
	1.3 Scope.....	
	2. Applicable Documents.....	
	3. Tactical Digital System Requirements.....	
	3.1 General.....	
	3.2 Program Description.....	
pspc_t3	3.3 Functional Description.....	
	3.4 Detailed Functional Requirements.....	
	3.5 Adaptation.....	
	4. Quality Assurance Provisions.....	
	4.1 General.....	
	4.2 Test Requirements.....	
	4.3 Acceptance Test Requirements.....	
	5. Preparation for Delivery.....	
	6. Notes.....	
	Appendixes	
	A. Applicable Documents.....	
	B. Glossary.....	
	C. Mathematical Analysis.....	
	D. Miscellaneous Items.....	

pspc_t4	LIST OF FIGURES	
	Figure	Page
pspc_t5	LIST OF TABLES	
	Table	Page
pspc_hd@		@spec heading@

Block ID	Block
pspc_t6	PROGRAM PERFORMANCE SPECIFICATION FOR
pspc_nm2@	@program name@
pspc_t7	1. Scope
pspc_t8	1.1 Purpose
pspc_t9	1.2 Mission
pspc_t10	1.3 Scope
pspc_t11	1.3.1 Identification
pspc_t12	1.3.2 Functional Summary
pspc_t13	2. Applicable Documents
pspc_t14	3. Tactical Digital System Requirements
pspc_t15	3.1 General
pspc_t16	3.2 Program Description
pspc_t17	3.2.1 General Description
pspc_t18	3.2.2 Peripheral Equipment Identification
pspc_t	3.2.3 Interface Identification
pspc_t20	3.3 Functional Description
pspc_t21	3.3.1 Equipment Descriptions
pspc_t22	3.3.2 Digital Processor Input/Output Utilization Table
pspc_t23	3.3.3 Digital Processor Interface Block Diagram
pspc_t24	3.3.4 Program Interfaces
pspc_t25	3.3.5 Function Description
pspc_t26	3.4 Detailed Functional Requirements
pspc_t27	3.4.n Introduction
pspc_t28	3.4.n.1 Inputs
pspc_t29	3.4.n.2 Processing
pspc_t30	3.4.n.3 Outputs

Block ID	Block
pspc_t31	3.4.n.4 Special Requirements
pspc_t32	3.5 Adaptation
pspc_t33	4. Quality Assurance Provisions
pspc_t34	4.1 General
pspc_t35	4.2 Test Requirements
pspc_t36	4.3 Acceptance Test Requirements
pspc_t37	5. Preparation for Delivery
pspc_t38	6. Notes
pspc_t39	Appendix A. Applicable Documents
pspc_t40	Appendix B. Glossary
pspc_t41	Appendix C. Mathematical Analysis
pspc_t42	Appendix D. Miscellaneous Items

3.AP.SPS.1.2.3 SOD Template

The SOD template is chosen by the specifict ion specialist module when [spec_type]=sod. The template is derived from the skeleton System Operational Design Document specified in appendix G of [SAM rqmt]. The outline generated by the specialist module will be identical to that skeleton specification with the addition of the actual values for the data derived from product characteristics.

Block ID	Block
sspc_date@	@spec date@
sspc_t1	SYSTEM OPERATIONAL DESIGN DOCUMENT FOR
sspc_nm1@	@program name@
sspc_t2	Prepared by
sspc_prep@	@preparer@

TABLE OF CONTENTS

Block ID	Block
sspc_t4	LIST OF FIGURES
Figure	Page
sspc_t5	LIST OF TABLES
Table	Page
sspc_hd@	@spec heading@
sspc_t6	SYSTEM OPERATIONAL DESIGN DOCUMENT FOR
sspc_nm2@	@program name@
sspc_t7	1. Introduction
sspc_t8	1.1 Purpose
sspc_t9	1.2 Mission
sspc_t10	1.3 Scope
sspc_t11	1.3.1 Identification
sspc_t12	1.3.2 Summary
sspc_t13	1.4 Concept of Operations
sspc_t14	1.5 Operational Program Design Concept
sspc_t15	1.5.1 Program Construction
sspc_t16	1.5.2 Program Capacities
sspc_t17	1.5.3 Console Modes and Service Arrays
sspc_t18	2. Applicable Documents
sspc_spcs@	@mil specs@
sspc_stds@	@mil standards@
sspc_t19	3. Operational Program Design
sspc_t20	3.1 General
sspc_t21	3.2 Program Support and Control Functions
sspc_t22	3.2.1 Program Support Functions
sspc_t23	3.2.2 Program Control
sspc_t24	3.2.3 Central Stores and Service Routines

Block ID		Block
sspc_t25	3.3	Operator Function Support
sspc_t26	3.3.1	Data Readout Implementation
sspc_t27	3.3.2	Operator Action Button Implementation
sspc_t28	3.3.3	Symbology Implementation
sspc_t29	3.3.4	Console Mode and Service Array Implementation
sspc_t30	3.4	Operator's Function Program Design
sspc_t31	3.4.1	Input Operations
sspc_t32	3.4.2	User Operations
sspc_t33	4.	System Equipment Operation
sspc_t34	4.1	General
sspc_t35	4.2	Combat Direction System Equipment
sspc_t36	4.3	Weapons Systems Equipment
sspc_t37	4.4	Peripheral Systems Equipment
sspc_t38	5.	Compatibility
sspc_t39	5.1	General
sspc_t40	5.2	Peripheral System Interface
sspc_t41	5.3	Operator Interface
sspc_t42	5.4	Intersystem On-Line Interface
sspc_t43	5.5	I/O Utilization Table
sspc_t44	5.6	Equipment Arrangement
sspc_t45	6.	Constraints
sspc_t46	6.1	General
sspc_t47	7	Program Design Budget
sspc_48	Appendix A.	Applicable Documents
sspc_48	Appendix B.	Glossary

3.AP.SPS.1.2.4 FOD Template

The FOD template is chosen by the specification specialist module when [spec_type]=fod. The template is derived from the skeleton Functional Operational Design Document specified in appendix H of [SAM rqmt]. The outline generated by the specialist module will be identical to that skeleton specification with the addition of the actual values for the data derived from product characteristics.

Block ID	Block
fspc_date@	@spec date@
fspc_t1	FUNCTIONAL OPERATIONAL DESIGN DOCUMENT FOR
fspc_nm1@	@program name@
fspc_t2	Prepared by
fspc_prep@	@preparer@

TABLE OF CONTENTS

fspc_t4 LIST OF FIGURES

Figure — Page

fspc_t5 LIST OF TABLES

Table — Page

Block ID		Block
fspc_hd@		@spec heading@
fspc_t6		FUNCTIONAL OPERATIONAL DESIGN DOCUMENT FOR
fspc_nm2@		@program name@
fspc_t7	1.	Introduction
fspc_t8	1.1	Purpose
fspc_t9	1.2	Function Requirement
fspc_t10	1.3	Scope
fspc_t11	1.3.1	Identification
fspc_t12	1.3.2	Summary
fspc_t13	1.4	Operational Programs
fspc_t14	2.	Applicable Documents
fspc_spcs@		@mil specs@
fspc_stds@		@mil standards@
fspc_t15	3.	Operational Design Components
fspc_t16	3.1	General
fspc_t17	3.2	Operator Actions
fspc_t18	3.2.1	Variable Action Button Allocation
fspc_t19	3.2.2	Fixed Action Button Allocation
fspc_t20	3.2.3	Number Entry Data Allocation
fspc_t21	3.2.4	General Purpose Action Codes
fspc_t22	3.2.5	Color Coding
fspc_t23	3.3	Action Data Processing
fspc_t24	3.3.1	Algorithms Implemented
fspc_t25	3.3.2	Communication Processing
fspc_t26	3.3.3	Display Processing
fspc_t27	3.4	Console Modes And Arrays

Block ID		Block
fspc_t28	3.4.1	Console Mode
fspc_t29	3.4.2	Console Arrays
fspc_t30	4.	Operator Function Sequence
fspc_t31	4.1	General
fspc_t32	4.2	Action Sequences
fspc_t33	4.2.1	Alerts
fspc_t34	4.2.2	Updates
fspc_t35	4.2.3	Communication Action
fspc_t36	4.3	Operator Monitor Function
fspc_t37	4.3.1	Tactical Displays
fspc_t38	4.3.2	Digital Displays
fspc_t39	4.3.3	Communication Guard
fspc_t40	5.	Test and Simulation Scenarios
fspc_t41	5.1	General
fspc_t42	5.2	Non-real-time Tests
fspc_t43	5.3	Real-time Tests
fspc_t44	5.4	Non-real-time Simulation
fspc_t45	5.5	Real-time Simulation
	Appendix A.	Applicable Documents
	Appendix B.	Glossary

3.AP.SPS.1.3 Local Dictionary

Data item	Definition
[edit_object]	a data item that conveys an editing action to be performed on a product building block of the specification object
[obj_id]	the identification of the object that represents the product being produced through the facilities of this

	specialist module; the identification is composed of [prod type] and [package_id]
[package_id]	the project identification and version identification of the acquisition package
[prod_type]	the type of product being produced by this specialist module; in this case the value of [prod_type] is "specification"
[read_id]	the identification of the specification object to be read from auxiliary storage
[spec_char]	the product characteristics needed by the specification specialist module to generate the specification outline
[spec_type]	the type of specification to be produced for the acquisition package; allowable values are: typea, pps, fod, or sod
@mil specs@	a list of the military specifications that are applicable to this procurement
@mil standards@	a list of the military standards that are applicable to this procurement
@preparer@	the name and address of the activity that is preparing the specification
@program name@	the name of the program for which the subject of this software acquisition is being procured; used for PPS, FOD, SOD
@spec date@	the publication date of the specification
@spec heading@	data used as a heading on each page of the body of the specification

@system name@ — the name of the embedded computer system for which the subject of this software acquisition is being procured; used for type A specification

%generated% — the status of the specification object has been set to "generated", i.e., the product characteristics necessary for generating the outline of the appropriate specification have been acquired and the specification outline has been generated

%incomplete% — the status of the specification object has been set to "incomplete", i.e., the specification object has been instantiated, but the acquisition of those product characteristics necessary for generating the outline of the appropriate specification has not been completed

%null% — an instance of a specification object for the current context does not exist

3.AP.SPS.1.4 Information Hidden

1. How the specification object is represented and stored.

2. The implementation of actions on the specification object by the specification specialist module.

3.AP.SPS.2 Design Support

3.AP.SPS.2.1 Interface Assumptions

3.AP.SPS.2.2 Design Issues

3.AP.SPS.2.3 Implementation/Configuration Information

3.AP.SPS.2.4 References

None.

3.AP.SWS Statement of Work Specialist (SWS) Module

The statement of work specialist module supports the creation of a statement of work for an acquisition package. The specialist module uses a template to assemble a statement of work outline. The template supplies both the initial structure and the initial content of the statement of work outline. The content of the outline is provided from literal text strings and from information derived from product characteristics. In the latter case, the template guides the specialist module in acquiring the information on product characteristics. The specialist module acquires further information as it becomes available to add, delete, and modify the text used to form the statement of work.

3.AP.SWS.1 Function Definition

3.AP.SWS.1.1 Actions

The statement of work specialist module operates as a process that performs actions when presented with a stimulus in the form of new or modified data items. These actions may result in a change or refinement to the statement of work object and/or a change to the statement of work status.

Action	Condition	Data Item	Response
+cr_sow+	%null%	[obj_id]	%incomplete%

Establishes a statement of work object by creating a statement of work appropriate to the user's requirements. The statement of work object is identified by [obj_id].

+gen_sow+	%incomplete%	[sow_char] [obj_id]	%generated%

Refines the statement of work object identified by [obj_id] by generating the statement of work outline. The specialist module generates the initial statement of work outline by assembling the product building blocks sequentially from the statement of work template. When it encounters a product building block that requires derivation of information from the product characteristics the specialist module acquires the needed data item and performs that function.

+mod_sow+	%generated%	[edit_object] [obj_id]	%incomplete% or %generated%

Refines the generated statement of work object identified by [obj_id] by acquiring one or more data items to set or change corresponding elements of the statement of work object. If a data item changes the value of a

product characteristic, the specialist module responds with %incomplete% to force regeneration of those portions of the outline that depend on the product characteristic whose value has changed. When no data items are available, the statement of work specialist module waits for one or more to be made available.

+cancel_sow+ NOT %null% [obj_id] %null%
The statement of work object identified by [obj_id] is deleted.

+print_sow+ NOT %null% [obj_id]
An image of the statement of work object identified by [obj_id] is printed.

+display_sow+ NOT %null% [obj_id]
An image of the statement of work object identified by [obj_id] is displayed.

+write_sow+ NOT %null% [obj_id]
A copy of the statement of work object identified by [obj_id] is transferred to the location in auxiliary storage addressed by the identification of the object. If a prior copy of the object had been made, it is deleted when the current copy is successfully completed.

+read_sow+ [read_id] %incomplete% or
[obj_id] %generated%
The copy of the statement of work object at a specified location in auxiliary storage is read by the statement of work specialist module. The location from which the object is read may be specified as either the current context or another context. In the former case, the effect is to read the most recently saved version of the statement of work object; in the latter case, the effect is to read a saved copy of a statement of work object from another acquisition package. The object that is read becomes the statement of work object identified by [obj_id] of the current context replacing the statement of work object which may have existed prior to the invocation of this action.

3.AP.SWS.1.2 Statement of Work Document Template

The template used by the statement of work specialist module to generate a statement of work outline is described in this section. The statement of work template guides the specialist module in generating a statement of work outline and in making modifications to the statement of work object in response to editing actions. A template is composed of uniquely identified product building blocks. Certain of the product building blocks contain literal text strings and will appear in the generated outline as they are shown in the template. Others contain data items bracketed with "@". These data items are derived from product characteristics acquired by the specialist module while generating the outline. The identifiers of blocks containing derived information are denoted with a suffix of "@".

The template is derived from the skeleton statement of work specified in appendix B of [SAM rqmt]. The outline generated by the specialist module will be identical to that skeleton statement of work with the addition of the actual values for the data derived from product characteristics.

Block Id	Block
sow_date@	@sow date@
sow_t1	STATEMENT OF WORK FOR
sow_nm1@	@program name@
sow_t2	Prepared by
sow_prep@	@preparer@

TABLE OF CONTENTS

	Section/Paragraph	Page
	1. Scope.....	
	2. Applicable Documents.....	
	2.1 Military Specifications.....	
	2.2 Military Standards.....	
	2.3 Other Publications.....	
sow_t3	3. Requirements.....	
	3.1 Computer Program Performance Requirements.....	
	3.2 Computer Program Design Requirements.....	
	3.3 Computer Program Production.....	
	3.4 Computer Program Operation.....	
	3.5 Program Test.....	

Block Id	Block
	3.6 Quality Assurance.....
sow_t3	3.7 Configuration Management.....
cont'd	3.8 Software Management Control.....
	3.9 Work Breakdown Structure.....
sow_hd@	@heading@
sow_t4	STATEMENT OF WORK FOR
sow_nm2@	@program name@
sow_t5	1. Scope.
sow_t6	2. Applicable Documents The following documents of the issue in effect on the date of solicitation form a part of this SOW to the extent specified herein.
sow_t7	2.1 Military Specifications
sow_spcs@	@mil specs@
sow_t8	2.2 Military Standards
sow_stds@	@mil standards@
sow_t9	2.3 Other Publications
sow_t10	3. Requirements
sow_t11	3.1 Computer Program Performance Requirements. The contractor shall determine the detailed program performance requirements for all software as specified in subsection 5.1 of MIL-STD-1679.
sow_t12	3.2 Computer Program Design Requirements. The contractor shall develop the detailed program design requirements in accordance with subsection 5.2 of MIL-STD-1679.
sow_t13	3.3 Computer Program Production. The contractor shall adhere to the detailed program design requirements as approved by the Government, and the System Specification in producing all computer programs. The contractor shall also use chief programmer teams and conform to the requirements of subsection 5.5 of MIL-STD-1679.
sow_t14	3.4 Computer Program Operation. The contractor shall determine the procedures for the operation of the defense system software in accordance with subsection 5.7 of MIL-STD-1679.

Block Id	Block
sow_t15	3.5 Program Test. The contractor shall determine the scope of tests required to ensure that the program being developed meets all specified technical, operational, and performance requirements and the acceptance criteria. The contractor shall be responsible for accomplishing all development testing. Testing shall be performed in accordance with requirements of subsection 5.8 of MIL-STD-1679, "Program Testing", unless otherwise specified below. Informal testing shall meet the following requirements: ° Tests shall be monitored primarily by contractor personnel and shall be subject to informal monitoring by the Government or its representative. ° The development plan shall be part of the TEMP or TEP. ° The tests shall constitute contractor internal milestones and informal project milestones. Formal testing shall meet the following requirements: ° The test shall constitute an official project milestone. ° The test shall be officially witnessed by the Government during its performance and shall be conducted in accordance with previously approved test specifications and procedures. ° All items that affect the test or that are used in the test, including hardware or software, must be certified before test. ° Tests shall be subsequently audited and reviewed by Government Quality Assurance (QA).
sow_t16	3.5.1 Program Unit Tests. Each lowest compilable unit will undergo the following tests as a minimum: a. Peer review b. Error-free compilation c. Exercise of logical execution paths d. Analysis of data flow monitoring, results of assignment, and exchange statements e. Validation of intended function Upon completion of unit testing, the software unit shall be incorporated under library control.
sow_t17	3.5.2 Module Tests. As specified in paragraph 5.8.1 of MIL-STD-1679.
sow_t18	3.5.3 Subprogram Tests. As specified in paragraph 5.8.2 of MIL-STD-1679.
sow_t19	3.5.4 Program Performance Tests. As specified in paragraph 5.8.3 of MIL-STD-1679.
sow_t20	3.5.5 Systems(s) Integration Test. System(s) integration testing involves the testing of software-software and software-hardware interfaces as subsystems are integrated into a larger system (or as one system in integrated with another). The contractor shall plan for and demonstrate progress against the plan to the Government during system integration test. Specific integration milestones shall be identified and scheduled.

Block Id	Block
sow_t20 cont'd	The Government shall be kept advised of the test schedules so that a designated Government representative can witness these tests. These tests shall be adequate to determine compliance with the applicable technical, operational, and performance requirements. As a minimum, system integration testing shall be performed to: a. Verify the total man-machine interface b. Validate system initiation, data entries via peripheral devices, program loading, restarting, and the monitoring and controlling of system operation from display consoles and other control stations as applicable c. Verify the interfacing of all equipment specified in the system requirements. d. Verify the capability of the program to satisfy all applicable system and program performance requirements e. Verify the capability of the system to handle properly and survive erroneous inputs f. Verify inter - and intrasystem message formats and interfaces
sow_t21	3.5.6 Software System Performance Test. Software system performance testing is formal and represents the final level of Development Test and Evaluation (DT&E) that is performed for the project. The contractor shall schedule, and the Government shall witness, a software system performance test to certify that the hardware and software represent the system as defined in the System Specification and that the QA provisions specified in Section 4 of the System Specification have been satisfied. As a minimum, software system performance testing shall be performed to: a. Verify the total man-machine interface b. Validate system initiation, data entries via peripheral devices, program loading, restarting, and the monitoring and controlling of system operation from display consoles and other stations as applicable c. Verify the interfacing of all equipment specified in the system requirements d. Verify the capability of the program to satisfy all applicable system, program performance, and QA requirements e. Verify the capability of the system to handle erroneous inputs properly and to survive them f. Verify inter - and intrasystem message formats and interfaces g. Verify system timings and specified constraints h. Verify constraints specified in this SOW.
sow_t22	3.6 Quality Assurance. The contractor shall implement a software quality assurance program in accordance with subsection 5.9 of MIL-STD-1679.
sow_t23	3.7 Configuration Management. The contractor shall develop and implement a software configuration management program in accordance with paragraphs 5.5.4 and 5.11 of MIL-STD-1679, and subsections 1.3, 3.0, 5.1 and Appendices I, VIII, IX, X, XII, XIV, and XV of MIL-STD-483, except as otherwise noted below in regard to configuration identification. Where conflicts arise between these standards, MIL-STD-1679 will

Block Id	Block
sow_t23 cont'd	govern. The contractor shall ensure that software CM procedures are integrated with other CM procedures addressing the total system.
sow_t24	3.7.1 Configuration Identification
sow_t25	3.7.1.1 Formal Baselines. The formal baselines required for the program are defined as follows:
sow_fbd@	° The Functional Baseline is determined by the @FB determinant@ and is under the configuration control of the Government.
sow_abd@	° The Allocated Baseline is determined by the @AB determinant@. The Allocated Baseline shall be under Government control.
sow_dbd@	° The Developmental Baseline is dynamic and is initially determined by the @DB determinant@. The @DB secondary determinants@, the final deliverable version of the program, all descriptive documentation, and the user manuals are also components of the Developmental Baseline and are added to the baseline as they are approved or accepted. As programs are written and pass minimum acceptance criteria, they shall be added to the Developmental Baseline under libary control. In its final configuration the Developmental Baseline shall constitute the software product baseline. The Developmental Baseline shall be under contractor control until final acceptance by the Government as the product baseline.
sow_pbd@	° The Product Baseline is determined by complete updated documentation that has been verified at PCA to reflect accurately the fully tested and accepted computer programs. This includes the final @PB determinant@, and all descriptive documentation and user manuals.
sow_t26	3.8 Software Manage- t Control. The cont ll implement a managment system for the software de fort that is acceptable to the procuring agency. ` trol shall conform to the requirements of subsecti -STD-1679 except as otherwise specified below.
sow_t27	3.8.1 R The r shall include formal and informal software reviews i. development schedule as described in succeeding paragraphs hese reviews can be incorporated with appropriate hardware reviews of a similar nature.
sow_t28	3.8.1.1 Formal Reviews. Formal reviews are those specific reviews designated by title in MIL-STD-1521A. These include the technical design reviews and audits for computer programs as follows. The Periodic Status Review is included as a formal review.

Block Id	Block
sow_srr@	3.8.1.1.1 System Requirements Review. The contractor shall hold a System Requirements Review (SRR) during the Requirements Definition activity to present the preliminary System Specification following functional analysis and preliminary requirements allocation. The contractor shall distribute a copy of the preliminary System Specification to the procuring agency for review at least @SRR prereview@ days before the SRR. All comments and questions arising from this review shall be returned to the contractor no later then @SRR prereview reply@ days before the SRR. The SRR shall be conducted in accordance with MIL-STD-1521A. The contractor shall answer the questions and comments generated by the procuring agency and shall make any required modifications to the System Specification.
sow_sdr@	3.8.1.1.2 System Design Review The contractor shall hold a System Design Review (SDR) for the purpose of reviewing and approving the final System Specification. The contractor shall distribute a copy of the System Specification to the procuring agency for review at least @SDR prereview@ before the SDR. All comments and questions arising from this review shall be returned to the contractor no later than @SDR prereview reply@ before the SDR. The SDR shall be conducted in accordance with MIL-STD-1521A. The contractor shall answer the questions and comments generated by the procuring agency and shall make any required modifications to the System Specification. The Preliminary Program Performance Specification (PPS) will be presented at the SDR.
sow_pdr@	3.8.1.1.3 Preliminary Design Review The contractor shall hold a Preliminary Design Review (PDR) for the purpose of reviewing and approving the final PPS. The contractor shall distribute a copy of the PPS to the procuring agency for review at least @PDR prereview@ before the PDR. All comments and questions arising from this review shall be returned to the contractor no later than @PDR prereview reply@ before the PDR. The PDR shall be conducted in accordance with MIL-STD-1521A. The contractor shall answer the questions and comments generated by the procuring agency and shall make any required modifications to the PPS. The preliminary Interface Design Specification (IDS), the preliminary Test Plan (TP), and the preliminary Program Design Specification (PDS) shall be presented at the PDR for procuring agency review and comment.
sow_cdr@	3.8.1.1.4 Critical Design Review The contractor shall hold a Critical Design Review (CDR) for the purpose of reviewing and approving the PDS, TP, and final IDS. The contractor shall distribute a copy of the PDS, TP, and IDS to the procuring agency for review at least @CDR prereview@ before the CDR. All comments and questions arising from this review shall be returned to the contractor no later than @CDR prereview reply@ before the CDR. The CDR shall be conducted in accordance with MIL-STD-1521A. The contractor shall answer the questions and comments generated by

Block Id	Block
sow_cdr@	the procuring agency and shall make any required modifications to the PDS, TP, and IDS.
sow_t29	3.8.1.1.5 Functional Configuration Audit A Functional Configuration Audit (FCA) shall be conducted to determine whether the CPCI has satisfied all requirements of the CPCI PPS. The FCA shall be conducted according to MIL-STD-1521A.
sow_t30	3.8.1.1.6 Physical Configuration Audit A Physical Configuration Audit (PCA) shall be conducted to determine whether the documentation accurately reflects the as-built computer programs. The conduct of a PCA is governed by MIL-STD-1521A.
sow_t31	3.8.1.1.7 Formal Qualification Review The contractor shall hold a Formal Qualification Review (FQR) for the purpose of reviewing the performance of the CPCI(s) as determined through test to verify that the CPCI(s) complies with its Program Performance Specifications and System Specification. On completion of FQR, the CPCI(s) shall be Government certified. The FQR shall be conducted in accordance with MIL-STD-1521A.
sow_t32	3.8.1.1.8 Periodic Software Project Status Reviews The contractor shall schedule monthly project status reviews throughout the contract period. These reviews will be attended by management personnel from the procuring agency, the contractor, and the subcontractor(s). Senior technical personnel shall attend if the contractor deems their presence to be required.
sow_t33	3.8.1.2 Informal Reviews The contractor shall conduct informal reviews throughout the software development cycle. These reviews are held for the purpose of domonstrating to the procuring agency that the software development and documentation are proceeding according to the approved specifications. Informal reviews may be held to present the results of analysis in answer to a procuring agency question or action item from a formal review. These reviews and demonstations do not require formal, deliverable supporting documentation; however, information as to their goals and a means of evaluating their performance shall be made available to the procuring agency before any such review. In-process reviews are informal technical reviews that are held to review the test specifications and procedures. They shall also be held to review the results of the structural walkthroughs of major segments of the software and to demonstrate progress during testing. Any discrepancies noted during the review or demonstration shall be recorded as a Software Trouble Report or an action item. The disposition of these items shall be monitored and included in the monthly progress reports to the procuring agency.

Block Id	Block
sow_wbs@	3.9 Work Breakdown Structure The preliminary Work Breakdown Structure (WBS), figure @WBS figure #@, graphically portrays the schedule of work to be accomplished under this contract consistent with the scope of work defined in the System Specification and SOW. Using the WBS supplied, the contractor will develop at least two additional levels of WBS elements for the Contractor WBS (CWBS). The CWBS shall be included as part of the submitted proposal and shall be presented in sufficient detail to show the bidder's understanding of the system requirements, the components composing the system, and the tasks to be performed during the acquisition cycle. The CWBS shall be constructed so that the procuring agency can readily identify the structural hierarchy of each component of the software system. In addition to the operational software components, the CWBS shall include support software that must be developed or modified by the contractor, as well as Government-furnished software that must be modified. The successful bidder shall add levels to his CWBS, if any are specified by the Government as being necessary, within @CWBS delivery@ from award of contract. Any changes to the CWBS after that time must receive approval from the procuring agency's program office.

3.AP.SWS.1.3 Local Dictionary

Data item	Definition
[edit_object]	a data item that conveys an editing action to be performed on a product building block of the statement of work object
[obj_id]	the identification of the object that represents the product being produced through the facilities of this specialist module; the identification is composed of [prod type] and [package_id]
[package_id]	the project identification and version identification of the acquisition package

[prod_type] the type of product being produced by this specialist module; in this case the value of [prod_type] is "statement of work"

[read_id] the identification of the statement of work object to be read from auxiliary storage

[sow_char] the product characteristics needed by the statement of work specialist module to generate the statement of work outline

@AB determinant@ a list of the formal documents which comprise the Allocated Baseline for configuration management

@CDR prereview@ the number of days prior to Critical Design Review that the Program Design Specification, Test Plan, and Interface Design Specifications will be made available to the procuring agency by the contractor

@CDR prereview reply@ the number of days prior to Critical Design Review that the questions and comments arising from the review of the Program Design Specification, Test Plan, and Interface Design Specifications will be made available to the contractor by the procuring agency

@CWBS delivery@ the number of days following award of contract that the contractor shall add levels to the Contractor Work Breakdown Structure

@DB determinant@ the formal documents which comprise the initial Developmental Baseline for configuration management

@DB secondary determinants@ the formal documents which comprise the final Developmental Baseline for configuration management

@FB determinant@ the formal documents which comprise the Functional Baseline for configuration management

@mil specs@ a list of the military specifications that are applicable to this procurement

@mil standards@ a list of the military standards that are applicable to this procurement

@PB determinant@ the formal documents which comprise the Product Baseline for configuration management

@PDR prereview@ the number of days prior to Preliminary Design Review that the final Program Performance Specification will be made available to the procuring agency by the contractor

@PDR prereview reply@ the number of days prior to Preliminary Design Review that the questions and comments arising from the review of the final Program Performance Specification will be made available to the contractor by the procuring agency

@preparer@ the name and address of the activity that is preparing the statement of work

@program name@ the name of the program for which the subject of this software acquisition is being procured

@sow date@ the publication date of the statement of work

@sow heading@ data used as a heading on each page of the body of the statement of work

@SDR prereview@ the number of days prior to System Design Review that the final System Specification will be make available to the procuring agency by the contractor

@SDR prereview reply@ the number of days prior to System Design Review that the questions and comments arising from the review of the final System Specification will be made available to the

contractor by the procuring agency

@SRR prereview@ the number of days prior to System Requirements Review tnat the preliminary System Specification will be made available to the procuring agency by the contractor

@SRR prereview reply@ the number of days prior to System Requirements Review that the questions and comments arising from the review of the preliminary System Specification will be made available to the contractor by the procuring agency

@WBS figure #@ the figure number of the WBS figure in the statement of work

%generated% the status of the statement of work object has been set to "generated", i.e., the product characteristics necessary for generating the outline of the statement of work have been acquired and the statement of work outline has been generated

%incomplete% the status of the statement of work object has been set to "incomplete", i.e., the statement of work object has been instantiated, but the acquisition of those product characteristics necessary for generating the outline of the statement of work has not been completed

%null% an instance of a statement of work object for the current context does not exist

3.AP.SWS.1.4 Information Hidden

1. How the statement of work object is represented and stored.

2. The implementation of actions on the statement of work object by the statement of work specialist module.

3.AP.SWS.2 Design Support

3.AP.SWS.2.1 Interface Assumptions

3.AP.SWS.2.2 Design Issues

3.AP.SWS.2.3 Implementation/Configuration Information

3.AP.SWS.2.4 References

None.

3.AP.WBS Work Breakdown Structure Specialist (WBS) Module

The work breakdown structure specialist module supports the creation of a work breakdown structure for an acquisition package. The specialist module uses a template to assemble a work breakdown structure. The template supplies both the initial structure and the initial content of the work breakdown structure. The content of the work breakdown structure is provided from literal text strings and from information derived from product characteristics. In the latter case, the template guides the specialist module in acquiring the information on product characteristics. The specialist module acquires further information as it becomes available to add, delete, and modify the text used to form the work breakdown structure.

3.AP.WBS.1 Function Definition

3.AP.WBS.1.1 Actions

The work breakdown structure specialist module operates as a process that performs actions when presented with a stimulus in the form of new or modified data items. These actions may result in a change or refinement to the work breakdown structure object and/or a change to the work breakdown structure status.

Action	Condition	Data Item	Response
+cr_wbs+	%null%	[obj_id]	%incomplete%

Establishes a work breakdown structure object. The work breakdown structure object is identified by [obj_id].

+gen_wbs+	%incomplete%	[wbs_char] [obj_id]	%generated%

Refines the work breakdown structure object identified by [obj_id] by generating the work breakdown structure hierarchy. The specialist module generates the initial work breakdown structure by assembling the product building blocks sequentially from the work breakdown structure template. When it encounters a product building block that requires derivation of information from the product characteristics the specialist module acquires the needed data item and performs that function.

+mod_wbs+	%generated%	[edit_object] [obj_id]	%incomplete% or %generated%

Refines the generated work breakdown structure object identified by [obj_id] by acquiring one or more data items to set or change elements of the work breakdown structure object. If a data item changes the value of a

product characteristic, the specialist module responds with %incomplete% to force regeneration of those portions of the outline that depend on the product characteristic whose value has changed. When no data items are available, the work breakdown structure specialist module waits for one or more to be made available.

+cancel_wbs+ NOT %null% [obj_id] %null%
The work breakdown structure object identified by [obj_id] is deleted.

+print_wbs+ NOT %null% [obj_id]
An image of the work breakdown structure object identified by [obj_id] is printed.

+display_wbs+ NOT %null% [obj_id]
An image of the work breakdown structure object identified by [obj_id] is displayed.

+write_wbs+ NOT %null% [obj_id]
A copy of the work breakdown structure object identified by [obj_id] is transferred to the location in auxiliary storage addressed by the identification of the object. If a prior copy of the object had been made, it is deleted when the current copy is successfully completed.

+read_wbs+ [read_id] %incomplete% or
[obj_id] %generated%
The copy of the work breakdown structure object at a specified location in auxiliary storage is read by the work breakdown structure specialist module. The location from which the object is read may be specified as either the current context or another context. In the former case, the effect is to read the most recently saved version of the work breakdown structure object; in the latter case, the effect is to read a saved copy of a work breakdown structure object from another acquisition package. The object that is read becomes the work breakdown structure object identified by [obj_id] of the current context replacing the work breakdown structure object which may have existed prior to the invocation of this action.

3.AP.WBS.1.2 Work Breakdown Structure Document Template

The template used by the work breakdown structure specialist module to generate a work breakdown structure is described in this section. The work breakdown structure template guides the specialist module in generating a work breakdown structure hierarchy and in making modifications to the work breakdown structure object in response to editing actions. The template is composed of uniquely identified product building blocks and their hierarchical relationships with each other. Certain of the product building blocks contain literal text strings and will appear in the generated outline as they are shown in the template. Others contain data items bracketed with "@". These data items are derived from product characteristics acquired by the specialist module while generating the work breakdown structure hierarchy. The identifiers of blocks containing derived information are denoted with a suffix of "@".

The template is derived from the skeleton work breakdown structure specified in appendix D of [SAM rqmt]. The hierarchy generated by the specialist module will be identical to that skeleton work breakdown structure with the addition of the actual values for the data derived from product characteristics.

Block Id	Block
wbs_nm@	@program name@
wbs_t1	SOFTWARE DEVELOPMENT
wbs_sq1@	@seqno@
wbs_t2	REQUIREMENTS ANALYSIS
wbs_sq2@	@seqno@01
wbs_lst2@	@subsystem list@
wbs_t3	PROGRAM PERFORMANCE REQUIREMENTS
wbs_sq3@	@seqno@02
wbs_lst3@	@subsystem list@
wbs_t4	PROGRAM DESIGN REQUIREMENTS
wbs_sq4@	@seqno@03

Block Id	Block
wbs_lst4@	@subsystem list@
wbs_t5	PROGRAM PRODUCTION
wbs_sq5@	@seqno@04
wbs_lst5@	@subsystem list@
wbs_t6	PROGRAM TEST
wbs_sq6@	@seqno@05
wbs_ls61	01 - Program Unit Tests
wbs_ls62	02 - Module Tests
wbs_ls63	03 - Subprogram Tests
wbs_ls64	04 - Program Performance Tests
wbs_ls65	05 - System(s) Integration Test
wbs_ls66	06 - Software System Performance Test
wbs_t7	PROJECT CONTROL
wbs_sq7@	@seqno@06
wbs_ls71	01 - Administration
wbs_ls72	02 - Quality Assurance
wbs_ls73	03 - Configuration Management
wbs_ls74	04 - Software Management Control
wbs_fg@	Figure @WBS figure #@

3.AP.WBS.1.3 Local Dictionary

Data item	Definition
[edit_object]	a data item that conveys an editing action to be performed on a product building block of the work breakdown structure object
[obj_id]	the identification of the object that represents the product being produced through the facilities of this specialist module; the identification is composed of [prod type] and [package_id]
[package_id]	the project identification and version identification of the acquisition package
[prod_type]	the type of product being produced by this specialist module; in this case the value of [prod_type] is "work breakdown structure"
[read_id]	the identification of the work breakdown structure object to be read from auxiliary storage
[wbs_char]	the product characteristics needed by the work breakdown structure specialist module to generate the work breakdown structure outline
@program name@	the name of the program for which the subject of this software acquisition is being procured
@seqno@	the first level work package number upon which all lower level work package numbers in the work breakdown structure hierarchy are based
@subsystem list@	a list of the software subsystems such that each is the subject of a separate set of requirements, design, and production activities; each element of the list consists of a subsystem name and a subsystem work package number

@WBS figure #@ — the figure number of the WBS figure in the work breakdown structure

%generated% — the status of the work breakdown structure object has been set to "generated", i.e., the product characteristics necessary for generating the outline of the work breakdown structure have been acquired and the work breakdown structure outline has been generated

%incomplete% — the status of the work breakdown structure object has been set to "incomplete", i.e., the work breakdown structure object has been instantiated, but the acquisition of those product characteristics necessary for generating the outline of the work breakdown structure has not been completed

%null% — an instance of a work breakdown structure object for the current context does not exist

3.AP.WBS.1.4 Information Hidden

1. How the work breakdown structure object is represented and stored.

2. The implementation of actions on the work breakdown structure object by the work breakdown structure specialist module.

3.AP.WBS.2 Design Support

3.AP.WBS.2.1 Interface Assumptions

3.AP.WBS.2.2 Design Issues

3.AP.WBS.2.3 Implementation/Configuration Information

3.AP.WBS.2.4 References

None.


DATE
ILME